POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Modellazione e ottimizzazione della pianificazione della didattica

Supervisore:

Prof. Cataldo Basile

Prof. Renato Ferrero

Candidato:

Francesco Lasagno

Anno Accademico 2021/2022
Torino

Ringraziamenti

Questa Tesi rappresenta per me la conclusione di un percorso di 5 anni intensi, fatto di alti e bassi, che mi ha permesso di crescere sia dal punto di vista accademico che soprattutto come persona.

Desidero quindi ringraziare tutte le persone che ho incontrato in questi anni e mi hanno permesso di diventare la persona che sono oggi.

Un ringraziamento particolare ovviamente va agli amici ed ai colleghi per tutti i momenti condivisi e per il supporto dimostratomi in questi anni.

Infine ringrazio la mia famiglia, ed in particolare mia mamma, mio papà e mio fratello per esserci sempre stati soprattutto negli ultimi mesi molto impegnativi per me.

Francesco

Indice

Immagini

Capitolo 1

Introduzione

La Tesi si pone come obiettivo di risolvere il problema la pianificazione dell'orario dei collegi di ICM (Informatica, Cinema e Meccatronica) ed ETF (Elettronica, Telecomunicazioni e Fisica) del Politecnico di Torino, modellandolo come un problema di ottimizzazione.

La pianificazione dell'orario è un problema difficile da affrontare poiché coinvolge differenti tipi di vincoli. Esistono limiti strutturali, quali aule e laboratori disponibili in quantità limitata e con capienze note, dipendenze date dai piani di studio, ad esempio l'impossibilità di sovrapporre insegnamenti obbligatori per un dato orientamento, vincoli dettati dalla disponibilità dei docenti che possono essere coinvolti in più insegnamenti, e vincoli dettati dalle necessità degli studenti di seguire troppe ore di lezione consecutive. Si è dovuto considerare anche che i docenti che possono avere impegni istituzionali o tenere lezione in insegnamenti in Master e corsi di laurea di altri collegi che comportano indisponibilità nell'arco della settimana.

Alcuni di questi vincoli sono stati ritenuti forti (*hard constraint*). Ad esempio, non è possibile pianificare un orario che usi più aule di quelle presenti in ateneo, non è possibile che un docente insegni contemporaneamente per più insegnamenti e non è possibile sovrapporre insegnamenti obbligatori all'interno dello stesso orientamento se questi sono allocati per il medesimo anno accademico.

Altri vincoli sono stati considerati deboli (*soft constraint*), soddisfare o non soddisfare i soft constraint introduce penalità o bonus a seconda del tipo. Ad esempio si pensi agli insegnamenti classificati come crediti liberi all'interno di un orientamento, se questi si sovrapponessero ad altri insegnamenti dello

stesso orientamento in un'allocazione questa non dovrebbe essere considerata non valida, ma bisognerebbe preferirne un'altra in cui tale sovrapposizione non è presente.

L'obiettivo della Tesi è stato sviluppare un software che supporti nella pianificazione dell'orario usando l'ottimizzazione matematica. Il problema è stato modellato richiedendo di soddisfare tutti gli hard constraint e massimizzare una funzione obiettivo costruita a partire da tutte i soft constraint. In altre parole, l'output di tale software è la versione migliore possibile dell'orario che rispetti tutti i vincoli.

Per raggiungere questo obiettivo è stato effettuato uno studio approfondito delle tecnologie e della letteratura informatica fatto per trovare i modelli di ottimizzazione e gli approcci migliori in grado di rappresentare correttamente le due tipologie di vincoli e che consentano di classificare le soluzioni in funzione del soddisfacimento o della violazione dei soft constraint. Sono stati individuati due problemi validi per quello in esame: "*Maximum satisfiability problem*" (MaxSMT) e "*Integer linear programming*" (ILP). In seguito sono stati scelti gli strumenti, i solver, in grado di risolvere tali problemi: Z3 per MaxSMT e IBM CPLEX per ILP.

I dati relativi all'analisi dei requisiti sono stati acquisiti mediante interviste con i responsabili dell'orario volte a chiarificare tutte le variabili in gioco e ad evidenziare eventuali casi limite di interesse.

Tramite l'analisi dei dati acquisiti si è ricavato un modello UML[1] rappresentante nell'interezza il problema; si è inoltre creato un modello UML specifico dei dati riguardanti gli insegnamenti di cui allocare l'orario, in quanto rappresentano la principale sorgente di dati del software. La progettazione di alto livello si è conclusa con la generazione degli schemi di validazione delle diverse sorgenti di informazioni per uniformare e automatizzare la lettura dei dati da parte del programma.

Successivamente, sulla modellazione si è passati dai requisiti di alto livello alle variabili e alle equazioni definite prima in un ambiente matematico e successivamente a livello di codice compatibilmente con le API messe a disposizione dai software utilizzati. Sono stati implementati due differenti modelli:

- Modello a matrice: enfatizza il concetto di unità di lezione allocata ad un

[1]Unified Modeling Language.

dato insegnamento e ad un dato docente in una data aula modellandolo tramite una matrice multidimensionale

- Modello a slot: viene posta l'enfasi sullo slot di lezione e sul giorno e la fascia oraria in cui questo viene allocato.

In entrambi i modelli i soft constraint vengono associati ad una funzione obiettivo da minimizzare associando ai singoli termini di essa un valore superiore a 0 in caso di penalità, 0 in caso di vincolo rispettato o suggerimento non rispettato e un valore inferiore a 0 in caso di suggerimento rispettato.

La modellazione in ambiente matematico ha evidenziato i limiti del modello a matrice in quanto, sebbene la definizione dei vincoli elementari fosse semplice, la descrizione di quelli più complessi era impraticabile; ad esempio un vincolo indicante l'allocazione parallela di due lezioni appartenenti ad insegnamenti differenti produce un elevato numero di equazioni con il modello matriciale mentre immediata con il modello a slot. D'altro canto il secondo modello rende più complessa la gestione dei vincoli intesi a porre un limite superiore al numero di lezioni allocabili parallelamente in funzione del numero di aule di un certo tipo disponibili in un certo giorno o ad una certa fascia oraria.

L'implementazione del modello matematico in Python ha evidenziato ulteriormente la bontà del secondo modello in termini di risorse computazionali necessarie e di performance. Non è stato infatti possibile modellare la disponibilità di aule con il primo modello poiché la richiesta di risorse era troppo elevata; inoltre il software CPLEX, a differenza di Z3, permette un'esecuzione multi-thread del solver. La scelta è quindi ricaduta sulla modellizzazione come un problema di programmazione lineare con l'ausilio di CPLEX.

Durante la fase successiva della Tesi è stato scritto il codice in grado di raccogliere i dati dalle diverse fonti di input, di istanziare le variabili del modello e di modellare i vincoli e la funzione obiettivo. È stato implementato un ulteriore modulo del software che ha il compito di interfacciarsi con il solver CPLEX e reagire ogni qualvolta viene trovata una soluzione e salvarla in un database di appoggio per una futura analisi ed esportazione.

Questa tesi ha anche affrontato il problema della validazione dei dati sorgente e delle soluzioni. Ad una prima fase di acquisizione dei dati da varie fonti tramite interazioni con i relativi uffici, nonché tramite questionari somministrati ai docenti ed all'estrazione semi automatizzata dei dati autonomamente da fonti pubblicamente consultabili quali il sito del Politecnico,

è iniziata la fase di validazione dei dati in input.

Le prime esecuzioni del software hanno permesso di verificare la correttezza dei dati inseriti andando a verificare l'esistenza di soluzioni che soddisfacessero i soli hard constraint. Successivamente si è considerato il problema di ottimizzazione nella sua interezza a ricercare la soluzione ottima considerando anche i soft constraint.

L'esportazione delle soluzioni e una prima validazione di massima sono avvenute tramite un modulo dedicato all'export e una piccola applicazione web che tramite interfaccia grafica mostra il piano di allocazione settimanale per orientamento o per singolo docente.

Nel Capitolo 2 sono introdotte le tecnologie e i software utilizzati nella Tesi, partendo da un'analisi della letteratura informatica e procedendo con l'introduzione di due differenti classi di problemi in grado di modellare il problema dell'allocazione dell'orario.

Nel Capitolo 3 vengono definiti tutti i requisiti del problema per poi essere suddivisi nelle due categorie di vincoli forti e deboli.

Nel Capitolo 4 è introdotta la progettazione di alto livello tramite un modello UML. Viene inoltre introdotto un secondo modello UML per uniformare le sorgenti di dato del software e vengono definiti

Nel Capitolo 5 vengono implementati a basso livello due modelli differenti per il problema in esame, sia in ambiente matematico che in ambiente di sviluppo andando a valutare pregi e difetti di entrambi.

Nel Capitolo 6 sono mostrate le modalità di utilizzo del software, sia in validazione che in ottimizzazione.

Infine nel Capitolo 7 viene descritto il reperimento dei dati di test e dei dati ufficiali per l'Anno Accedemtico 2022-2023, l'utilizzo del tool per la risoluzione del problema, l'esportazione dei risultati e la loro validazione.

Nelle conclusioni di Capitolo 8 compaiono alcuni spunti per una futura espansione del modello e di miglioramento del software oltre ai manuali d'uso del software che mostrano tutte le opzioni che è possibile configurare ed i vincoli che è possibile aggiungere oltre ad indicare come espandere lo strumento con nuove funzioni.

Capitolo 2

Background

Dopo una ricerca nella letteratura informatica di problemi noti simili a quello oggetto di Tesi si sono individuati due approcci differenti per affrontarlo: modellandolo come *Maximum Satisfiability problem* con l'ausilio del software Z3 e modellandolo come *Integer Linear Programming problem* con l'ausilio del software CPLEX[1]. Entrambi gli approcci sono presentati in questo Capitolo.

2.1 Problema Job-Shop Scheduling

Nella letteratura informatica il problema del Job-Shop Scheduling consiste nell'allocare risorse condivise e presenti in maniera limitata a diverse attività nel tempo[1, 2]. Il modello descrive un insieme di n jobs $\{J_i\}_{0 \leq i < n}$ costituiti ognuno da diverse Operazioni O da eseguire in ordine sequenzialie su m macchine $\{M_r\}_{0 < \leq r < m}$.

Esistono vincoli di precedenza tra i diversi Job tali per cui:

$$time(J_i) < time(J_j)$$

L'esecuzione di un'Operazione di un Job su una macchina ne prevede un uso esclusivo, mentre il tempo necessario al suo completamento viene chiamato l.

La funzione obiettivo del Job-Shop scheduling è di minimizzare il *makespan* L corrispondente al tempo necessario a completare tutti i jobs.

Viene definita schedulazione un'allocazione che permette di eseguire in un tempo L tutti i jobs. Il problema è NP-Completo.

[1]Il problema è NP-Completo

2.2 SAT e maxSAT

2.2.1 Logica proposizionale

Nella logica proposizionale le proposizioni sono costruite partendo da variabili booleane e composte usando solamente connettivi logici quali $\wedge, \vee, \neg$.
I SAT solver utilizzano un approccio chiamato *systematic search* in cui lo spazio di ricerca è un albero in cui ogni nodo rappresenta una singola variabile booleana e gli archi uscenti dal vertice la scelta *true* o *false* per quella variabile. Data una formula con n variabili, ci sono 2^n foglie nell'albero. Viene definito *model (modello)* un percorso dalla radice a una foglia tale per cui la formula risulti vera [3, 4].

La costruzione del modello avviene mediante l'algoritmo *DPLL*[2] usando tre operazioni: *decide*, *propagate* e *backtrack*. L'algoritmo sfrutta la possibilità di scrivere ogni proposizione logica in una equivalente forma detta *Forma Normale Congiuntiva CNF*[3].

Esempio 1. Una proposizione CNF valida è:

$$(A \vee \neg B \vee \neg C) \wedge (\neg D \vee E \vee F) \tag{2.1}$$

supponendo che tutte le variabili siano atomiche booleane. Per chiarezza si noti che ogni clausola corrisponde alla singola espressione tra parentesi. Una proposizione non CNF è:

$$\neg(B \vee C) \tag{2.2}$$

che può essere riscritta nella corrispondente forma CNF:

$$(\neg B) \wedge (\neg C) \tag{2.3}$$

L'operazione *decide* sceglie euristicamente una variabile atomica a cui non è ancora stato assegnato un valore e vi assegna true o false[4]. L'operazione *propagate* ricava le conseguenze della soluzione parziale usando alcune regole di deduzione, tra cui la principale si chiama *unit-clause rule* che asserisce "se

[2]Davis-Putnam-Logemann-Loveland: algoritmo di ricerca esaustiva, basato su backtracking, utilizzato per decidere la soddifabilità booleaa di formule di logica proposizionale CNF.

[3]Una formula CNF è una formula booleana tale per cui ogni clausola è una disgiunzione di letterali; un letterale è una variabile atomica o la negazione di una variabile atomica.

[4]Questa operazione è detta anche *branching* o *case splitting*.

tutti i letterali di una clausola tranne uno sono assegnati a false il letterale rimanente l è ancora indefinito, allora l'unica possibilità per la clausola per essere valuta true è quella di assegnare true a l".

Data la clausola $C := p \vee \neg q \vee \neg r$ e la soluzione parziale $M := \{p \rightarrow false, r \rightarrow true\}$, allora l'unico modo per C per essere true è assegnare a q false. Con una parziale valorizzazione M e una clausola C all'interno della CNF tale per cui tutti i letterali di di C sono assegnati a false, allora non è possibile estendere M ad un modello completo M' che soddisfa la formula data ma si è in *conflitto* e C è in una *clausola conflittuale*.

Un conflitto indica che alcune delle decisioni precedentemente prese non portano ad un assegnamento valido per la formula data e quindi l'algoritmo DPLL procede con un *backtrack* e prova ad usare un branch differente. Se è rilevato un conflitto e non è possibile effettuare *backtrack* su nessuna decisione si deduce che la formula è *unsatisfiable (non soddisfacibile)*.

2.2.2 Satisfiability Modulo Theories

La logica booleana introdotta non risulta sufficiente per modellare il problema del Job Shop Scheduling; per ovviare a questo si introduce un nuovo tipo di problema chiamato *Satisfiability Modulo Theories* (SMT); questo consiste nel verificare se una formula logica F è soddisfacibile secondo una derminata teoria con dei vincoli su F. Si dice che una formula F è soddisfacibile se esiste un'interpretazione tale per cui F è vera [4].

Esempio 2. La formula

$$a + b > 3 \wedge a < 0 \wedge b > 0 \tag{2.4}$$

è soddisfacibile secondo la teoria aritmetica poichè esiste l'interpretazione

$$a \mapsto -1, b \mapsto 5 \tag{2.5}$$

Il problema del Job-Shop Scheduling come detto necessita di disequazioni lineari per essere modellato, ed in particolare combinando un SAT solver con un solver SMT che supporti la teoria dell'aritmentica delle differenze[5]. L'aritmetica delle differenze è una branchia dell'aritmetica lineare dove i predicati sono ristretti alla forma $t - s \leq c$ dove t e s sono variabili mentre c

[5]Supportata da Z3.

è una costante numerica. Un problema SMT estende un problema di logica booleana includendo oltre alle proposizioni logiche anche predicati riferiti ad un insieme di variabili non binarie, come ad esempio disequazioni. Questo fatto permetterà anche di trasformare un problema di soddisfacibilità in un problema di ottimizzazione assegnando dei costi alle variabili.

2.2.3 Il solver Z3

Z3 è un solver SMT che sfrutta in sinergia l'analisi, la verifica e l'esecuzione simbolica del codice. Nella realizzazione del software oggetto di questa Tesi si è fatto uso delle API sviluppate per Python.

Tra i vari tipi di proposizioni supportate da Z3 ve ne sono due di particolare rilevanza che sono state utilizzate nella realizzazione del modello: *Propositional Logic* e *Integer Linear Arithmetic*.

Esempio 3. Di seguito un esempio di *Propositional solving* in Z3 in cui il modello è rappresentato da 3 variabili atomiche booleane.

```
1  from z3 import *
2  Auto, Patente, Bicicletta = Bools("auto patente bicicletta")
3  s = Solver()
4  s.add(Or(
5      And(Patente, Auto), Bicicletta
6      ))
7  s.add(Not(
8      And(And(Patente,Auto), Bicicletta
9      )))
10 print(s.check())
11 print(s.model())
12
13 # output:
14 sat # s.check()
15 [bicicletta = False, auto = True, patente = True] # s.model()
```

Dopo l'allocazione delle variabili si procede con l'instanziazione del solver (riga 3) e con l'aggiunta di due proposizioni logiche (riga 4 e 7) che porta il solver a generare le seguenti asserzioni come constraint per il modello:

$$(Patente \wedge Auto) \vee Bicicletta$$

$$\neg((Patente \wedge Auto) \wedge Bicicletta)$$

Alla riga 10 viene quindi verificata la *soddisfacibilità*[6] del modello ed in caso di esito positivo viene estratta una soluzione (riga 11).
Z3 prevede anche l'utilizzo di proposizioni aritmetiche lineari sia nell'insieme dei numeri Interi che in quello dei Reali; nella realizzazione del modello si è fatto uso solamente di proposizioni aritmetiche con numeri Interi.

2.2.4 Problema MaxSAT con Z3

La definizione di problema MaxSAT è di minimizzare il numero di soft constraint violati. Assumendo che tutti i vincoli non siano pesati, ovvero di peso unitario, si definisce un problema MaxSAT con associati i predicati constraint $P_1, ..., P_n, F_1, ..., F_n$ dove i predicati P_* sono hard constraint mentre quelli F_* sono soft. L'obiettivo sarà quindi quello di trovare un'interpretazione tale per cui sia soddisfatti tutti i P_* ed al contempo sia soddisfatto il numero maggiore possibile di F_*[5].

Esempio 4. Dato il problema MaxSAT con $P, F_1, .., F_5$ dove i primi 4 soft constraint non possono essere soddisfatti insieme all'hard constraint P. Scritto formalmente:

$$\mathrm{A} : \mathrm{P}, \mathrm{F}_1, \mathrm{F}_2, \mathrm{F}_3, \mathrm{F}_4, \mathrm{F}_5 \tag{2.6}$$

Si definisce quindi il prblema "allentato" A':

$$\mathrm{A}' : \mathrm{P}, \mathrm{F}_2 \vee \mathrm{F}_1, \mathrm{F}_3 \vee (\mathrm{F}_2 \wedge \mathrm{F}_1), \mathrm{F}_4 \vee (\mathrm{F}_3 \wedge (\mathrm{F}_3 \wedge (\mathrm{F}_2 \wedge \mathrm{F}_1))), \mathrm{F}_5 \tag{2.7}$$

A questo punto la risoluzione di A' corrisponde a trovare una soluzione ottima per A. Si definisce quindi il *costo(M,A)* come numero di soft constraint di A che sono falsi nell'interpretazione M. Di conseguenza:

$$\forall \text{ model M di F, } \textit{costo(M,A)} = 1 + \textit{costo(M,A')} \tag{2.8}$$

La dimostrazione è disponibile in un articolo pubblicato dall'università di Stanford [6].

Esempio 5. Di seguito un esempio di definizione di un problema MaxSAT in Z3.

```
1 from z3 import *
2 Auto, Patente, Bicicletta = Bools("auto patente bicicletta")
3 s = Optimize()
```

[6]Un problema si definisce sat se ne esiste almeno una soluzione che soddisfa tutte le proposizioni.

```
s.add(Or(And(Patente, Auto), Bicicletta))
s.add(Not(And(Auto, Bicicletta)))

s.add_soft(Bicicletta,1)

print(s.check())
print(s.model())

# output:
sat # s.check()
[bicicletta = True, patente = True, auto = False] # s.model()

# se aggiungo anche s.add_soft(Auto,2)
[bicicletta = False, patente = True, auto = True] # s.model()
```

2.3 Integer Linear Programming

Un problema di programmazione intera vincolata è un problema ottimizzazione o ammissibilità in cui parte o tutte le variabili sono intere. Nella Tesi si utilizza un particolare caso di questo problema chiamato *Integer Linear Programming* (ILP) in cui i vincoli e la funzione obiettivo sono lineari. La complessità del problema risulta NP-completa.

2.3.1 Definizione

Un problema di ottimizzazione lineare ha come scopo trovare la soluzione che minimizza la funzione obiettivo f tra tutte le soluzioni X che soddisfano l'insime di m vincoli V per le n variabili intere del problema [7]:

$$\begin{gathered} min(max) f(x) \\ v_i(x) \leq, \geq o = b_i, \forall i = 1..m \\ x \in \mathrm{N}^n \end{gathered} \tag{2.9}$$

dove:

- $x = (x_1, x_2, .., x_n)$ è il vettore di n variabili intere che rappresenta una possbilie soluzione del problema
- $f(x)$ e tutte le $v_i(x)$ sono definite come $\mathrm{N}^n \rightarrow \mathrm{N}$

- $0 \leq i < m,\ b_i \in \mathrm{N}$.

La linearità del problema permette di definire f e tutti i vincoli v_i come funzioni lineari delle variabili di x:

$$\begin{gathered} f : c_1x_1 + c_2x_2 + .. + c_nx_n \\ v_i : a_{i1}x_1 + a_{i2}x_2 + .. + c_{in}x_n = b_i \\ oppure : \\ v_i : a_{i1}x_1 + a_{i2}x_2 + .. + c_{in}x_n = b_i \\ oppure : \\ v_i : a_{i1}x_1 + a_{i2}x_2 + .. + c_{in}x_n = b_i, 0 \leq i < m \end{gathered} \tag{2.10}$$

Una *soluzione ammissibile* di un problema di programmazione lineare è un vettore $x_@$ che soddisfa tutti i vincoli.

Una *soluzione ottima* di un problema di programmazione lineare è un vettore x_* che minimizza o massimizza la funzione obiettivo f.

L'insieme X di tutte le soluzioni ammissibili è detto *regione ammissibile*.

La risoluzione di un problema di programmazione lineare può ricadere in 3 casi:

- problema *innammissibile*: l'insieme X delle soluzioni ammissibili è vuoto

- problema *illimitato*: è possibile trovare soluzioni ammissibili che migliorano la funzione obiettivo all'infinito

- problema che *ammette soluzione ottima*: esiste almeno una soluzione x_* che minimizza o massimizza la funzione obiettivo; ne segue che il valore k definito come $f(x_*) = k$, appartiene ai numeri naturali.

Proprietà 1: Dato un problema di programmazione lineare, se il poliedro P delle soluzioni ammissibili è non vuoto e limitato, allora esiste almeno una soluzione ottima corrispondente con un vertica di P chiamato vertice di P ottimo.

2.3.2 Forma standard

Prima di procedere ad una manipolazione algebrica di un problema di programmazione lineare lo si trasforma nella *forma standard*:

$$\begin{gathered} \min z : c_1 x_1 + c_2 x_2 + .. + c_n x_n \\ \text{s.t.} a_{i1} x_1 + a_{i2} x_2 + .. + a_{in} x_n = b_i, 0 \leq m \\ x_i \in \mathrm{N}_+ \end{gathered} \tag{2.11}$$

dove:

- la funzione obiettivo è di minimo (si moltiplicano per -1 le funzioni di massimizzazione)
- tutte le variabili sono positive o nulle (si effettuano cambi di variabili)
- tutti i vincoli sono delle equazioni (si aggiunge una variabile positiva di slack per i vincoli $\leq$ e si sottrae una variabile positiva di surplus per i vincoli $\geq$
- i termini noti b_i sono tutti positivi o nulli (si moltiplicano per -1 i vincoli con termine noto negativo).

Questa trasformazione permette di risolvere il problema di programmazione lineare tramite un sistema di equazioni lineari.

2.3.3 Soluzioni di base

Un articolo di Zambelli [7] introduce un metodo per risolvere un sistema di equazioni lineari facendo uso del concetto di base di una matrice. Sia data una matrice $\mathrm{A} \in \mathrm{R}^{\mathrm{m}\times\mathrm{n}}$ di rango massimo. Una base di A è una sottomatrice quadrata di A di rango massimo definita come $\mathrm{B} \in \mathrm{R}^{\mathrm{m}\times\mathrm{m}}$ ottenuta scegliendo m colonne linearmente indipendenti della matrice A.

Da una rielaborazione del sistema lineare si può quindi scrivere:

$$Ax = b \Longrightarrow [B|F] \begin{bmatrix} x_B \\ x_F \end{bmatrix} = Bx_B + Fx_F = b \tag{2.12}$$

dove F è l'insieme delle colonne di A non appartenenti alla base, x_B il vettore delle variabili corrispondenti alle colonne nella base e x_F il vettore delle restanti variabili.

Data l'invertibilità di B possiamo esplicitare nella (2.12) x_B:

$$x_B = B^{-1}b + B^{-1}Fx_F \tag{2.13}$$

Al variare delle $(n - m)$ variabili fuori dalla base si ottengono differenti soluzioni $[x_B x_F]$ che risultanto ammissibili soltanto se tutte le loro componenti sono positive. Una soluzione al sistema $Ax = b$ si ottiene ponendo a 0 tutte le variabili fuori dalla base:

$$x = \begin{bmatrix} x_B \\ x_F \end{bmatrix} = \begin{bmatrix} B^{-1} \\ 0 \end{bmatrix} \tag{2.14}$$

Scelta una base B le soluzioni ottenute si dicono *soluzioni di base* con al più m variabili diverse da 0.

Caratterizzazione algebrica dei vertici di un politopo: dato un sistema di equazioni $Ax = b$ ed il corrispondente poliedro della regione ammissibile $P = \{x \in \mathcal{R}^n : Ax = b\}$, x è soluzione base del sistema $Ax = b \Leftrightarrow x$ è vertice di P.

2.3.4 Metodo e algoritmo del simplesso

Il metodo del simplesso permette di trovare la soluzione ottima di un problema di programmazione linerare partendo da una soluzione ammissibile sotto in modo iterativo generando ad ogni iterazione una nuova soluzione che migliora la precedente.

Di seguito vengono elencati i passaggi fondamentali del metodo, per una trattazione esaustiva fare riferimento all'articolo [7].

- Passaggio alla forma standard: il problema va riscritto nella forma standard come descritto nella (2.11), andando eventualmente a cambiare il verso della funzione obiettivo

$$\begin{aligned} &\min\ z = c^T x \\ &\text{t.c. } Ax = b, x \geq 0 \end{aligned} \tag{2.15}$$

- Scelta della base iniziale e passaggio alla forma canonica (o di tableau): viene scelta la base di partenza in modo che nei vincoli le variabili in base siano espresse unicamente in termini delle variabili non di base e

che la funzione obiettivo sia scritta in termini delle sole variabili non di base. In termini matriciali moltiplicando per B^{-1} la (2.12) si ottiene:

$$\begin{aligned} \min\ z &= c^T x \\ Ix_B + B^{-1}Fx_F &= B^{-1}b, x \geq 0 \end{aligned} \tag{2.16}$$

Con le sostituzioni $x_B = B^{-1}b - B^{-1}Fx_F$ e successivamente $\overline{\mathrm{F}} = B^{-1}F$, $\overline{\mathrm{c}}_{\mathrm{F}}^{\mathrm{T}} = c_F^T - c_B^T B^{-1}F$, $\overline{\mathrm{b}} = B^{-1}b$, e $z_B = c_B^T B^{-1}b$ si ottiene una definizione del problema equivalente all'originario dove z_B è il valore della funzione obiettivo per la soluzione di base $x_B = B^{-1}b, x_F = 0$ associata a B:

$$\begin{aligned} \min\ z &= \overline{\mathrm{c}}_F^T x_F + z_B \\ Ix_B + \overline{\mathrm{F}}x_F &= \overline{\mathrm{b}}, x \geq 0 \end{aligned} \tag{2.17}$$

- Scelta della variabile entrante per il cambio di base: permette di passare ad una soluzione ammissibile adiacente scambiando una variabile in base con una fuori base. La scelta della variabile da far entrare in base è determinata dalla ricerca di una nuova soluzione che migliori la funzione obiettivo. Partendo dalla defininizione della funzione obiettivo nella (2.17) (forma di tableau) si ha:

$$z = z_B + \overline{\mathrm{c}}_F^T x_F = \overline{\mathrm{c}}_{F_1} x_{F_1} + .. + \overline{\mathrm{c}}_{F_j} x_{F_j} + .. \tag{2.18}$$

da cui viene introdotta in base la variabile x_{F_j} che assumerà un valore positivo mentre le restanti variabili rimarranno a 0, da cui il nuovo costo della funzione obiettivo sarà:

$$z = z_B + \overline{\mathrm{c}}_{F_j} x_{F_j} \tag{2.19}$$

Il valore $\overline{\mathrm{c}}_{F_j}$ rappresenta quindi l'incremento marginale della funzione obiettivo all'aumentare di x_{F_j} e viene detto *costo ridotto*. In conclusione conviene far entrare in base una variabile fuori base con costo ridotto negativo ($\overline{\mathrm{c}}_{F_j} < 0$) tale da avere un miglioramento della funzione obiettivo (x_{F_j} è > 0).

- Scelta della variabile uscente: data x_j la variabile appena entrata i base e $\overline{\mathrm{a}}_{ij}$ il coefficiente della variabile x_j nel vincolo i-esimo, la variabile uscente sarà data dalla regola del quoziente minimo in quanto l'obiettivo è di massimizzare x_j senza violare i vincoli:

$$\min_{i:\overline{\mathrm{a}}_{ij}>0} : \left\{ \frac{\overline{\mathrm{b}}_i}{\overline{\mathrm{a}}_{ij}} \right\} \tag{2.20}$$

- Cambio base (pivot): la variabile entrante x_j deve prendere posto nella riga della variabile uscente x_k nella nuova base.

- Terminazione: il problema può terminare in una condizione di ottimalità una volta che tutti i costi ridotti sono non negativi. Il probema è invece illimitato qualora tutti i coefficienti della colonna corrispondente alla variabile entrante sono negativi o nulli.

L'applicazione del metodo del simplesso richiede la presenza di una soluzione ammissibile di base. La ricerca di tale soluzione è possibile tramite il metodo delle due fasi; per ulteriori dettagli si faccia riferimento a [7].

2.3.5 Il solver CPLEX

In accordo con l'articolo [8] estratto dalla documentazione ufficiale del software CPLEX, esso è in grado di risolvere, tra gli altri, anche problemi di programmazione lineare. L'utilizzo del software in questa modalità prevede la definizione di vincoli lineari come quelli definiti nell'equazione (2.11)[7].

Il software implementa tre algoritmi per risolvere problemi di programmazione lineare:

- Simplex Optimizer: prevede di muoversi lungo i bordi del poliedro per raggiungere i vertici rappresentanti una soluzione ammissibile per il problema, fino a raggiungere quello corrispondente alla soluzione ottima come mostrato in Figura 2.1.

- Dual-simplex Optimizer: la sua trattazione andrebbe al di fuori degli scopi della Tesi, si faccia riferimento a [8].

- Barrier Optimizer: utilizza un approccio diverso muovendosi non lungo i bordi della regione ammissimibile ma di muoversi al suo interno correggendo continuamente il suo percorso tramite un algoritmo predictor-corrector. La Figura 2.2 mostra graficamente il comportamento di questo algoritmo.

[7]In realtà CPLEX non accetta direttamente disuguaglianze del tipo $\leq$ o $\geq$, risulta tuttavia immediato il passaggio a $<$ o $>$.

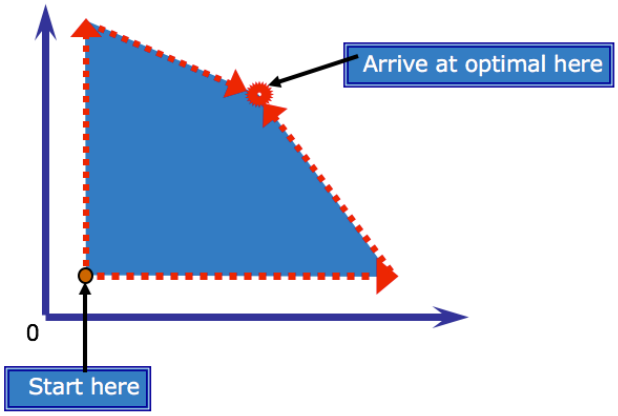
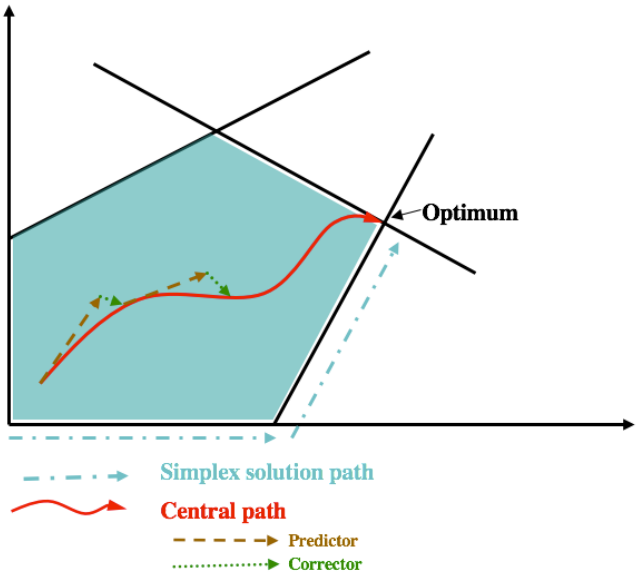


Figura 2.1: Algoritmo del Simplesso implementato in CPLEX.



[tb]

Figura 2.2: Algoritmo Barrier Otpimizer.

I vincoli logici definiti in termini di *if_then()*, *logical_and()* e *logical_or()* sono trasformati automaticamente in equazioni lineari con l'introduzione di ulteriori variabili in accordo con la documentazione ufficiale di CPLEX [9].

Esempio 6. Di seguito viene mostrato un semplice esempio di utilizzo del software CPLEX tramite le API per Python.

```
# istanziazione del modello
mdl = CpoModel("Modello")

# definizione delle variabili del modello
mat1 = mdl.integer_var_list(LEN*LEN, -9, 9, "mat")
matSlack = mdl.binary_var_list(LEN*LEN, "slack")

# aggiunta dei vincoli
for i in range(LEN):
    mdl.add(mdl.sum([mat1[i*LEN+j] for j in range(LEN)]) == 10)
mdl.add(if_then(mat1[i] == 8, matSlack[i] == 1) for i in range(LEN*LEN))
mdl.add(mdl.sum([(mat1[i] == 9) * 1 for i in range(LEN*LEN)]) <= 30)

# definizione della funzione obiettivo
mdl.minimize(mdl.sum(mat1[i] for i in range(LEN*LEN)))

# avvio del solver
msol = mdl.solve(TimeLimit=10)
```

Capitolo 3

Definizione del problema

La generazione dell'orario consiste nel ricavare una soluzione che permetta di svolgere le varie lezioni nell'arco della settimana tenendo conto della disponibilità di risorse limitata di risorse a disposizione in termini di numero di Aule, disponibilità dei Docenti a fare lezione, necessità di garantire agli studenti la possibilità di seguire determinati Insegnamenti lo stesso semestre (che quindi non potranno essere erogati contemporaneamente in aule differenti) e più in generale di soddisfare vari vincoli di natura diversa che verranno trattati in seguito nel dettaglio.

3.1 Esempi introduttivi

Vengono ora mostrati due esempi per entrare maggiormente nell'ottica del problema:

Esempio 7.

1. "si hanno 8 lezioni di 1,5h ciascuna che vanno erogate il Lunedì mattina a partire dalle 8:30 e devono concludersi entro le ore 16:00"
2. "vi sono 2 Aule distinte"
3. "le lezioni sono così ripartite tra tre diversi Insegnamenti: 3 lezioni (da qui in avanti chiamate Slot) sono di Geografia, 2 Slot (3 ore) sono di Storia e i rimanenti 3 Slot (4,5 ore) sono di Italiano"
4. "gli studenti che seguono il corso di Storia devono poter seguire anche il corso di Italiano"
5. "il Docente del Corso di Storia è il medesimo di quello del corso di Geografia"
6. "è noto che la soglia di attenzione degli studenti cali notevolmente dopo 3 ore di lezione della stessa materia".

Dai requisiti sopra esposti è possibile dedurre l'impossibilità di pianificare l'erogazione di tutti e tre gli Insegnamenti contemporaneamente (vi è un limite di due aule), non è valido un orario in cui le ore di Italiano e Storia vengano erogate in parallelo (gli studenti devono poter seguire entrambi gli Insegnamenti), discorso analogo anche per la coppia Storia e Geografia (il Docente che tiene le lezioni è il medesimo); infine le 4,5 ore di lezione di Geografia andranno divise poichè non è possibile avere più di 3 ore consecutive della stessa materia.

L'Esempio 7 corrisponde ad un problema di soddisfacilità che dovrà essere implementato su un dataset con una cardinalità dell'ordine di 600 Slot di lezione, 250 differenti Docenti, 20 Orientamenti composti da differenti Insegnamenti che non potranno essere erogati contemporaneamente (gli studenti che scelgono un Orientamento devono poterlo seguire interamente) tenendo conto dei limiti dettati dalle Aule messe a disposizione nonchè della loro capienza (risulta evidente l'impossibilità di erogare un Insegnamento con 300 studenti iscritti in un Aula con 50 postazioni).

Si introduce ora un problema di ottimizzazione:

Esempio 8.

1. "Il Docente del corso di Storia e Geografia preferisce terminare le sue lezioni con Storia perchè la ritiene meno impegnativa per i suoi studenti"
2. "Gli studenti del corso di Geografia principalmente non vivono vicino alla scuola e quindi potrebbero avere difficoltà ad essere puntuali alle 8:30".

L'Esempio 8 consente di ricavare differenti pianificazioni delle lezioni rispettando tutti i vincoli dell'Esempio 7, ma non tutte saranno ottime.

3.2 Requisiti

I requisiti costituiscono le asserzioni e le deduzioni di alto livello dedotti a seguito di interviste con i responsabili dell'orario necessari alla progettazione del modello:

1. "Vi sono Corsi di Laurea, Orientamenti, Insegnamenti, Slot[1], Fasce Orarie, Giorni, Locali, Docenti".

[1]Si intende la singola lezione indivisibile erogabile e può occupare 1.5 ore o un suo multiplo.

2. “I Corsi di Laurea sono di tipo Magistrale o Triennale”.
3. “Un Corso di Laurea prevede diversi Orientamenti[2]”.
4. “Un Orientamento propone diversi Insegnamenti[3]”.
5. “Un Orientamento è costituito da più anni: tre per gli Orientamenti di Corsi di Laurea Triennali e due per quelli Magistrali”.
6. “Un Orientamento ha *Insegnamenti obbligatori*, *Insegnamenti suggeriti*, *Insegnamenti a Scelta* molto correlati, a media correlazione e poco correlati per ogni anno[4]”.
7. “Un Orientamento contiene alcuni Insegnamenti che devono essere considerati per la consecutività[5]”.
8. “Un Insegnamento si tiene in uno o più semestri ed è associato ad un anno per ogni Orientamento”.
9. “Un Insegnamento è composto da uno o più Slot”.
10. “Per ogni Insegnamento viene definito un Template Orario[6]”.
11. “Un Insegnamento è associato a uno o più Docenti e uno di essi è titolare di esso[7]”.
12. “Un Insegnamento è frequentato da un certo numero di studenti”.

---

[2]Per ogni Corso di Laurea vi sono più Orientamenti.

[3]Un Insegnamento può essere presente in Orientamenti differenti.

[4]I vari Insegnamenti all’interno di un Orientamento non hanno la medesima importanza, in quanto non tutti sono obbligatori al conseguimento del titilo, inoltre alcuni di essi sono obbligatori ma in mutua esclusione tra loro. Si veda la Tabella 3.1 per una descrizione rigorosa della correlazione tra Insegnamenti all’interno di un determinato Orientamento

[5]Viene introdotto un limite sul numero massimo di ore di lezione giornaliere allocabili a tali Insegnamenti; l’appartenenza di un Insegnamento a tale gruppo dipende dall’*intensità* della relazione Orientamento-Insegnamento.

[6]Rappresenta la suddivisione desiderata dei vari Slot all’interno della settimana, specificando per ognuno di essi le risorse necessarie per la sua allocazione.

[7]Il Docente titolare non deve necessariamente essere presente in tutti gli Slot dell’Insegnamento; la sua figura è importante in quanto si occupa di definire il Template Orario per l’Insegnamento.

13. “Un Insegnamento ha un piano di allocazione settimanale fisso per l’intera durate del semestre”.

14. “Un Insegnamento fa riferimento ad un Collegio”.

15. “Uno Slot può essere una *Lezione*, un’*Esercitazione in Aula* o un’*Esercitazione in Laboratorio*”.

16. “I Locali possono essere *Aule* e *Laboratori*. Le Aule si suddividono a loro volta in *Aule attrezzate CA2*, *Aule attrezzate CA*, *Aule TableBox*, *Aule WallBox* e *Aula 5T*; mentre i laboratori sono: *LABINF*, *LAIB*, *LADISPE*, *LED*, *LED1*, *LED2*, *ACSLAB*”.

17. “Gli Slot vanno allocati in un tipo di Locale specifico”.

18. “Un Insegnamento può prevedere Slot in cui gli studenti sono divisi in squadre; quindi potranno essere allocati anche in parallelo”.

19. “Ad ogni Slot sono associati uno o più Docenti”.

20. “I Docenti possono fornire informazioni in merito al fatto che le loro lezioni all’interno di un Insegnamento sono limitate in un periodo specifico del semestre”.

21. “Ogni Slot si svolge in un giorno della settimana: dal Lunedì al Venerdì ed eventualmente il Sabato”.

22. “Ogni Slot dura 1,5 ore e si svolge in una fascia oraria: dalle 8:30 alle 19:00”.

23. “Le Aule hanno una *capienza* massima”. Fatto strettamente in relazione con il numero di studenti frequentanti un Insegnamento.

24. “Uno Slot è assegnato ad un *Locale*”.

25. “I Docenti assegnano *preferenze* circa gli slot orari in cui desiderano a fare lezione su una scala discreta di 3/5 valori differenti[8]”.

26. “I Docenti possono indicare un numero limitato di slot orari in cui sono impossibilitati a tenere una lezione[9]”.

[8]Informazioni non vincolanti al fine di trovare una soluzione valida.

[9]Informazioni vincolanti.

27. “I Docenti potrebbero avere ulteriori incompatibilità dovute a Insegnamenti erogati all’interno di altri Collegi”.
28. “Per ogni Insegnamento viene definito un *Piano di Allocazione settimanale*”.
29. “Due Aule si trovano ad una certa distanza”.
30. “Gli studenti scelgono un Orientamento e un piano di studi”.
31. “Due Insegnamenti possono essere in conflitto in modo vincolante o non vincolante”.
32. “Un Piano di Allocazione corrisponde all’orario settimanale di tutti gli Insegnamenti contenente tutti gli Slot che li costituiscono con le relative informazioni circa il Giorno, la Fascia oraria, i Docenti e il Locale assegnati per ognuno di essi”.
33. “Esistono dei vincoli tra Slot appartenenti allo stesso Insegnameto”.
34. “Esiste il piano di allocazione dell’anno precedente”.
35. “I Docenti possono esprimere una valutazione di gradimento dell’allocazione dell’anno precedente”.
36. “I Docenti titolari di un Insegnamento definiscono l’allocazione ideale degli Slot durante la settimana”.

3.3 Analisi dei requisiti

Dopo un’accurata analisi dei requisiti sopra esposti è stata fatta un’estrazione dei vincoli caratterizzanti il problema andandoli a suddividere in due macro categorie:

- *Hard Contraint*: ne fanno parte tutti i vincoli il cui rispetto è obbligatorio al fine di ottenere una soluzione valida
- *Soft Constraint*: ne fanno parte tutti i vincoli che non è obbligatorio rispettare per ottenere una soluzione valida; tuttavia l’osservazione o meno di questi permette di valutare la bontà di una soluzione mediante l’introduzione di penalità/bonus. La somma algebrica di tali punteggi consente di stabilire un criterio di ordinamento tra le soluzioni, consentendo di trovare la *soluzione ottima*.

3.3.1 Hard Constraint

Segue ora una caratterizzazione formale di tutti gli Hard Constraint introdotti nel modello.

- *Slot allocabili* solo in prefissati slot orari[10]: dal Lunedì al Venerdì dalle 8:30 alle 19:00 e opzionalmente il Sabato mattina dalle 8:30 alle 13:00.
- *Locali*:
  - Uno Slot va allocato in un Locale di capienza congrua alla cardinalità degli studenti frequentanti[11] l'Insegnamento.
  - Uno Slot va allocato in un Locale del *tipo* richiesto.
- *Slot*:
  - Per un determinato Insegnamento è possibile raggruppare all'interno dello stesso giorno un limite di 2 entità Slot[12] consecutive.
  - Alcuni Slot che devono essere allocati in un predeterminato slot orario.
  - Possono essere definite delle relazioni binarie Slot: *almenoUnGiornoDopo*, *subitoDopo*, *stessoGiorno*, *giornoDiverso* e *parallelo*.
- *Slot* di uno o più Insegnamenti all'interno dello stesso Orientamento:
  - Gli Slot di Insegnamenti obbligatori non devono mai essere allocati nello stesso slot orario.
  - Gli Slot di Insegnamenti suggeriti dello stesso Orientamento non devono mai essere allocati nello stesso slot settimanale.
  - Uno Slot di un Insegnamento suggerito e uno Slot di un Insegnamento obbligatorio non devono mai essere allocati nello stesso slot settimanale.

[10]Singola unità di tempo allocabile e corrisponde a 1 ora e mezza.

[11]Negli anni passati si è osservato come una parte degli studenti seppur sia iscritta all'Insegnamento non ne frequenta le lezioni; si considera una buona approssimazione assumere come numero di studenti frequentanti il numero di essi iscritto all'Insegnamento per la prima volta.

[12]Singola entry nel Template Orario di un dato Insegnamento.

  - Se un Insegnamento ha più di uno Slot allocato lo stesso giorno, nel caso i Docenti che svolgono lezione in tali Slot siano anche solo parzialmente in comune[13] e nel caso tali Slot abbiano come tipologia *"lezione"* allora necessariamente essi saranno allocati in maniera contigua.

- *Docente*:

  - Lo stesso Docente non può essere assegnato contemporaneamente a due Insegnamenti differenti (salvo richieste esplicite del Docente).
  - Un Docente non può avere allocati più di 4 Slot in un dato giorno (salvo limiti differenti da considerare puntualmente per docenti con molti slot da allocare).
  - Non è possibile allocare uno Slot tenuto da un Docente in uno slot settimanale in cui ha indicato incompatibilità.
  - Un Docente può esprimere una distanza minima/massima in termini di slot orari tra Slot a lui collegati ma appartenenti a Insegnamenti diversi.
  - Un Docente può richiedere l'uso dell'Orario dell'anno precedente.
  - Un Docente può non voler avere allocati in un medesimo giorno Slot appartenenti ad Insegnamenti differenti.

- *Studente* iscritto ad un Orientamento:

  - Non deve avere più di 5 slot orari consecutivi allocati in un determinato giorno tra gli Insegnamenti definiti obbligatori, suggeriti o obbligatori a scelta[14] per il suo Orientamento.
  - Non deve avere più di 6 slot orari allocati in un determinato giorno tra gli Insegnamenti definiti obbligatori, suggeriti o obbligatori a scelta.

[13] Ad uno Slot possono essere associati più Docenti.

[14] In un Orientamento è prassi comune offrire gruppi di Insegnamenti tra cui scegliere in mutua esclusione che risultano essere obbligatori; si pensi ad esempio al caso di un Insegnamento erogato in inglese e italiana.

	Obbligatorio	Obbligatorio a scelta	Suggerito	Tabella a scelta	Credito libero
Obbligatorio	hard const.	hard const.	hard const.	LV_4*k	LV_3*k
Obbligatorio a scelta	hard const.	hard const.	hard const.	LV_4*k	LV_3*k
Suggerito	hard const.	hard const.	LV_4*k	LV_3*k	LV_2*k
Tabella a scelta	LV_4*k	LV_4*k	LV_3*k	LV_2*k	LV_1*k
Credito libero	LV_3*k	LV_3*k	LV_2*k	LV_1*k	LV_0

Tabella 3.1: tipologia Insegnamenti in Orientamento.

3.3.2 Soft Constraint

Di seguito una caratterizzazione formale di tutti i Soft Constraint introdotti nel modello.

- *Slot*:
  - Possono essere definite delle relazioni binarie Slot: *almenoUnGiornoDopo*, *subitoDopo*, *stessoGiorno*, *giornoDiverso* e *parallelo*.
- *Slot* di uno o più Insegnamenti all'interno dello stesso Orientamento:
  - La sovrapposizione di Slot appartenenti a Insegnamenti differenti non trattata come hard constraint è soggetta a penalità variabili secondo la Tabella 3.1:

    Il calcolo della penalità deve tenere conto del fatto che due Insegnamenti possono essere presenti in molteplici Orientamenti e che la loro sovrapposizione può causare un disagio ad una popolazione più o meno vasta di studenti a seconda del numero di questi frequentanti gli Insegnamenti coinvolti.
    Viene quindi introdotta la seguente formula per calcolare le penalità derivanti dalla sovrapposizione di Slot di Insegnamenti presenti nello stesso Orientamento:

$$penalty = \text{MAX(penalty\_sovr)} + \text{SUM}(penalty\_sovr * delta^{15}),$$
$$delta = 0.1$$

  - In certi Orientamenti sono presenti coppie o terne di Insegnamenti tra cui lo studente deve scegliere solo uno di questi Insegnamenti e questi vanno considerati di tipo obbligatorio all'interno dell'Orientamento. Considerare puntualmente gli Insegnamenti appartenenti ai singoli gruppi permette di allocare eventualmente in parallelo gli Insegnamenti appartenenti allo stesso gruppo.

- *Docente*:
  - Un Docente può esprimere parere favorevole o contrario circa la possibilità di fare lezione in uno slot settimanale.
  - Un Docente può esprimere una distanza minima/massima in termini di slot orari tra Slot a lui collegati ma appartenenti a Insegnamenti diversi nello stesso giorno.
  - Un Docente può esprimere la preferenza a mantenere l'Orario dell'anno precedente.
  - Un Docente può indicare di preferire di non aver allocati lo stesso giorno Slot appartenenti ad Insegnament differenti.
  - É necessario un meccanismo di bilanciamento per evitare di assecondare soltanto le richieste di alcuni Docenti a scapito di altri.

- *Studente* iscritto ad un Orientamento:
  - Se uno studente di un orientamento ha 4 slot orari di lezione consecutivi tra gli Insegnamenti definiti *core* per l'Orientamento allora viene assegnata una penalità LV_1 al Piano Allocazione.
  - Se uno studente di un orientamento ha 5 slot orari di lezione consecutivi tra gli Insegnamenti definiti *core* per l'Orientamento allora viene assegnata una penalità LV_2 al Piano Allocazione.
  - Se nell'orario di uno studente di un orientamento nella stessa giornata è presente un intervallo di 2 slot orari tra Slot appartenenti ad Insegnamenti definiti *core* per l'Orientamento allora viene assegnata una penalità LV_1 al Piano Allocazione.
  - Se nella stessa giornata è presente un intervallo di almeno 3 slot orari tra Slot appartenenti ad Insegnamenti definiti *core* per l'Orientamento.
  - Se nella stessa giornata sono presenti due intervalli tra Slot appartenenti ad Insegnamenti definiti *core* per l'Orientamento allora viene assegnata una penalità LV_6 al Piano Allocazione.

Capitolo 4

Progettazione

4.1 Modelli adottati

Di seguito la rappresentazione rigorosa del modello in UML in grado di rispettare tutti i requisiti esposti nel capitolo precedente. Successivamente vengono definiti i formati scelti per l'input dei dati all'interno del software.

4.1.1 Modello UML globale

Data la vastità del modello l'analisi del modello è suddivisa in più porzioni:

- Corso di Laurea, Orientamento e Insegnamento:

  La porzione di modello in Figura 4.1 evidenzia i molteplici Orientamenti all'interno di un dato Corso di Laurea; inoltre gli studenti si iscrivono ad un dato Orientamento e di conseguenza ai diversi Insegnamenti di afferenza a questo. Inoltre ogni Orientamento è costituito da più Insegnamenti e per ognuno di questi vi sono alcuni dati molto rilevanti tra cui:

  – ID_INC: rappresenta l'identificativo dell'Insegnamento all'interno del modello.
  – *listCodIns*: un dato Insegnamento viene identificato nei vari piani studi pubblicamente disponibili con codici differenti a seconda dell'Orientamento in cui è presentato[1].

[1]Poichè lo stesso Insegnamento può essere molto numeroso è possibile esistano differenti

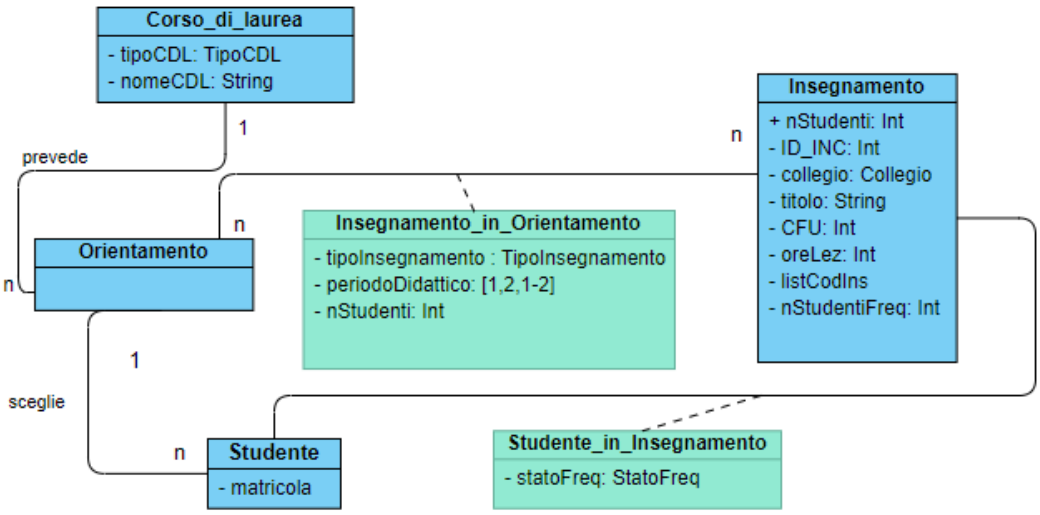


Figura 4.1: Modello UML che descrive l'Insegnamento.

– *nStudenti* e *nStudentiFreq*: i loro valori sono conseguenza della relazione *Studente_in_Insegnamento*.

La relazione *Insegnamento_in_Orientamento* contiene al suo interno l'attributo *nStudenti* che permetterebbe di dosare meglio le penalità a seconda degli Orientamenti a cui si riferiscono, tuttavia non è stato possible ricavare tali dati e quindi al momento risulta inutilizzata.

I tipi associati agli attributi sono riportati qui di seguito:

```
1 tipoCdl in ['1', 'Z'] # il Corso di Laurea è rispettivamente
    Triennale o Magistrale
2 tipoInsegnamento in [Obbligatorio, ObbligatorioAScelta, CreditoLibero
    , TabellaAScelta]
3 statoFreq in [Attivo, NonFrequentante, FrequenzaConvalidata] # a
    causa dei dati a disposizione si utilizza direttamente il numero
    di Studenti al primo anno di frequenza nel software
```

- Insegnamento e Docenti:

Alfabetiche per lo stesso (quindi diversi ID_INC) e ciò può portare ad avere anche casi in

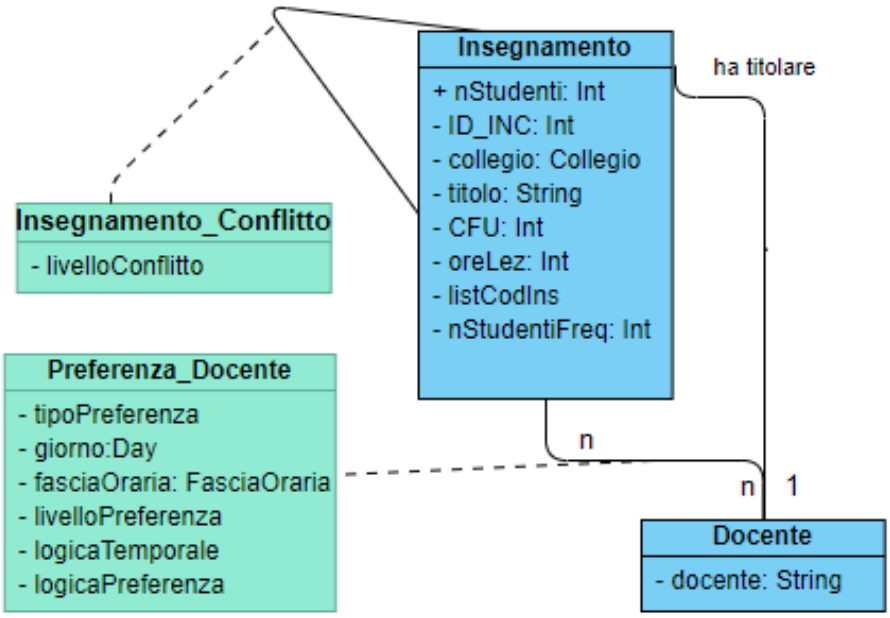


Figura 4.2: Modello UML che descrive gli Insegnamenti del Docente.

La parte di modello in Figura 4.2 riporta le relazioni tra Docenti e Insegnamenti:

- *Insegnamento_Conflitto* si occupa della modellazione dei vincoli dovuti all'appartenenza di un Insegnamento ad uno o più Orientamenti.
- *Preferenza_Docente*: data la disomogeneità di possibili constraint modellabili questa parte verrà approfondita successivamente andando ad esaminare i dati richiesti in input dal software.
- *Insegnamento_ha_titolare* risulta utile in quanto soltanto tale Docente ha il compito di fornire i dati di cui ai punti precedenti.

I tipi associati agli attributi precedentemente non introdotti sono riportati qui di seguito:

cui lo stesso *codIns* si riferisca a più ID_INC.

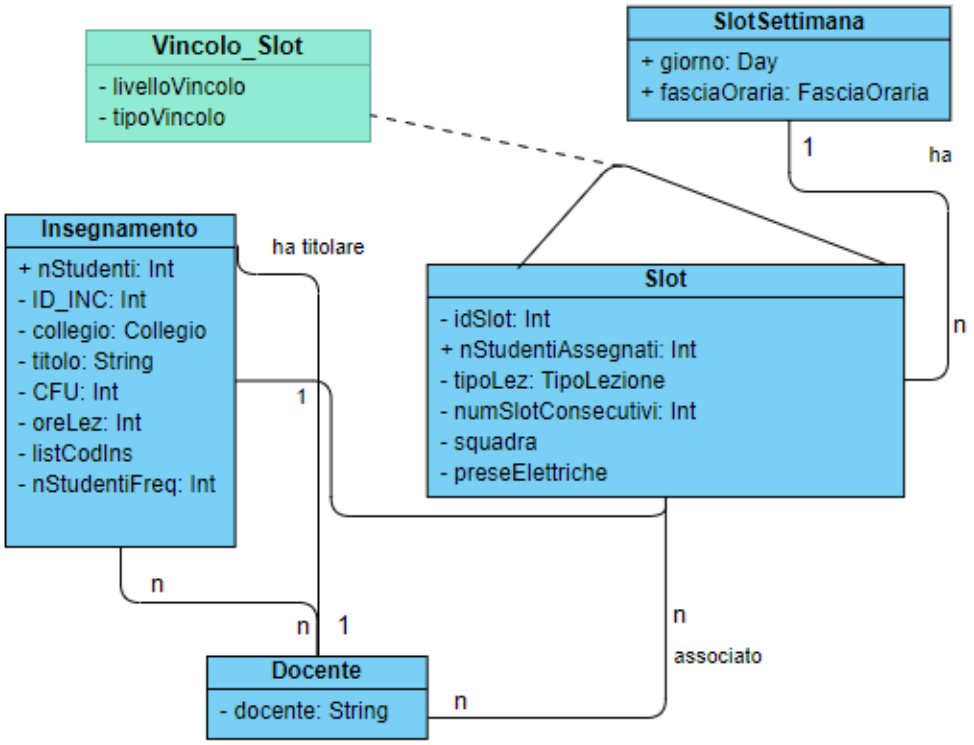


Figura 4.3: Modello UML che descrive gli Slot di un Insegnamento.

```
livelloConflitto in [LVH, LV1, LV2, LV3, LV4, LV5] # LVH corrisponde
    al modellare un hard constraint, mentre gli altri valori
    corrispondo a degli interi progressivi
giorno in [Lun, Mar, Mer, Gio, Ven, Sab, 'Indifferente']
fasciaOraria in [8.30-10.00, 10.00-11.30, 11.30-13.00, 13.00-14.30,
    14.30-16.00, 16.00-17.30, 17.30-19.00, 'Indifferente']
tipoPreferenza in ['incompatibilita', 'maxSlotPerDay', '
    insegnamentiSovrapponibili', 'preferenzeAnnoPrecedente', '
    orariLezioniDocente', 'distanzaMinima', 'distanzaMassima', '
    insegnamentiDiversiSameDay']
livelloPreferenza in [LVH, LV1, LV2, LV3, LV4, LV5]
logicaTemporale in ['primaDi', 'dopoDi', 'ugualeA']
logicaPreferenza in ['favorevole', 'contrario']
```

- Slot Insegnamento e Docenti:

  Dal modello in Figura 4.3 si evince che un Insegnamento è costituito da

uno o più Slot.
Per ogni Slot esistono alcune relazioni e alcuni attributi di interesse:

- *Docente_in_Slot*: in un dato Slot è possibile che più Docenti facciano lezione[2]. È necessario elencare puntualmente tutti i Docenti che dovranno tenere tale lezione poichè sarà necessario evitare sovrapposizioni con altri Slot in cui è presente anche solo uno dei Docenti elencati[3].
- *Slot_ha_SlotSettimana*: ogni Slot deve essere identificato temporalmente all'interno nella settimana; inoltre l'attributo *numSlotConsecutivi* ha il compito di considerare più slot orari come un'unica entità che deve necessariamente essere allocata in un unico blocco contiguo[4].
- *squadra*: attributo opzionale che se presente in due diversi Slot dello stesso Insegnamento e con valore diverso consente al software di allocarli anche parallelamente[5].
- *Vincolo_in_Slot*: viene concesso ampio margine gestionale ai titolari degli Insegnamenti consentendo loro di personalizzare con un elevato livello di dettaglio i vari Slot che compongono il singolo Insegnamento; si tratta di relazioni binarie il cui significato è autoesplicativo dai nomi stessi[6].

I tipi associati agli attributi precedentemente non introdotti sono riportati qui di seguito:

[2]Potrebbe trattarsi di compresenze oppure più semplicemente di lezioni tenute in settimane o periodi differenti del semestre da Docenti differenti.

[3]Salvo richieste esplicite dei Docenti trattate puntualmente.

[4]Seppur sia possibile istanziare due Slot di durata inferiore ed un vincolo di contiguità per ottenere il medesimo risultato che si ottiene istanziando un unico Slot ove *numSlotConsecutivi* corrisponde alla somma degli altri due; è preferibile istanziare un unico Slot al fine di dimezzare le variabili introdotte nel modello, come verrà illustrato in seguito.

[5]Di norma due Slot afferenti allo stesso Insegnamento non possono essere allocati in parallelo poichè ogni studente deve poter seguire tutte le lezioni dello stesso.

[6]Il software è corredato di un linguaggio booleano elementare che consente di esprimere espressioni più o meno complesse mediante gli operatori *OR*, *AND* e *NOT*.

```
giorno in [Lun, Mar, Mer, Gio, Ven, Sab, 'Indifferente']
fasciaOraria in [8.30-10.00, 10.00-11.30, 11.30-13.00, 13.00-14.30,
    14.30-16.00, 16.00-17.30, 17.30-19.00, 'Indifferente']
tipoLezione in [L, EA, EL, L_EA] # rispettivamente Lezione,
    Esercitazione Aula, Esercitazione Laboratorio, Lezione o
    Esercitazione in Aula.
livelloVincolo in [LVH, LV1, LV2, LV3, LV4, LV5]
tipoVincolo in ['almenoUnGiornoDopo', 'subitoDopo', 'stessoGiorno', '
    giornoDiverso', 'parallelo']
squadra in ['Squadra 1', 'Squadra 2', 'Squadra 3', 'Squadra 4'] #
    opzionale, se non specificato si intende che lo Slot non sia
    riferito ad una specifica squadra
```

- Slot e Locali:

  L'entità Locale ed i suoi figli riportati in Figura 4.4 contengono le informazioni relative agli spazi fisici disponibili e assegnabili:

  – *preseElettriche* e *tipoLocale* permettono di assegnare al singolo Slot un Locale che rispetti i requisiti.

  – *capienza* permette classificare i Locali in differenti gruppi a seconda della capienza[7].

  La relazione *Locale_Distanza* permette di quantificare la distanza tra Locali al fine di minimizzare gli spostamenti tra Locali differenti al variare degli Slot di lezione erogati.

  I tipi associati agli attributi precedentemente non introdotti sono riportati qui di seguito:

```
preseElettriche in ['Si', 'No', 'Preferibile']
tipoLocale in ['Aula', 'Aula multimediale', 'Aula attrezzata CA2', '
    Aula attrezzata CA', 'Aula TableBox', 'Aula WallBox', 'LABINF', '
    LAIB', 'LADISPE', 'LED', 'LED1', 'LED2', 'ACSLAB', 'Aula 5T']
capienza in ['Piccola', 'Media', 'MedioGrande', 'Grande', '
    NonDisponibile']
```

[7]Seppur valorizzabile, l'informazione sulla capienza di un'Aula trova riscontro pratico solamente per le Aule e i locali ad essa affini in quanto un dato laboratorio non esiste in istanze multiple con capienza differente.

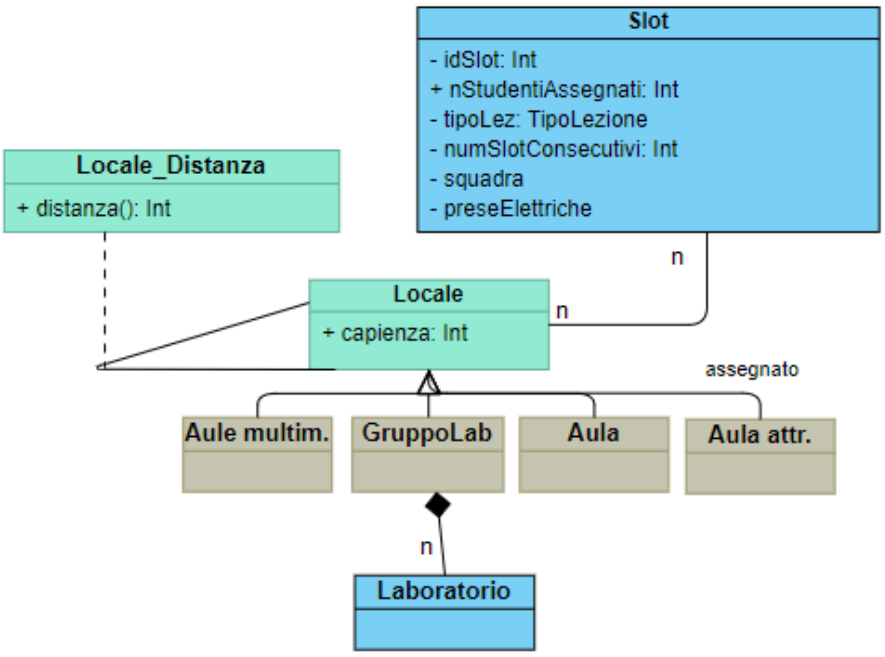


Figura 4.4: Modello UML che descrive i tipi di Locale per gli Slot.

4.1.2 Modello UML Template Orario

Il software è progettato per ricevere separatemente un'istanza del modello UML che rapresenta un *Template Orario* per ogni insegnamento che deve essere allocato. Un Template Orario rappresenta i requisti espressi Docenti titolari di un Insegnamento in termini di organizzazione settimanale degli slot orari da erogare, informazioni sui Docenti che terranno la lezione in tali slot orari, i Locali richiesti per erogare specifiche lezioni, squadre a cui si riferiscono i diversi Slot.

- Template Orario e SlotTemplate:

  Larga parte del modello UML corrisponde a quello globale in quanto questa è una sua estensione, di seguito vengono spiegate nel dettaglio alcune scelte fatte. In particolare il *TemplateOrario* in Figura 4.5 rappresenta la modellazione dell'orario dell'Insegnamento ed è una collezione di *SlotTemplate* che lo definiscono, tra i suoi attributi vi sono:

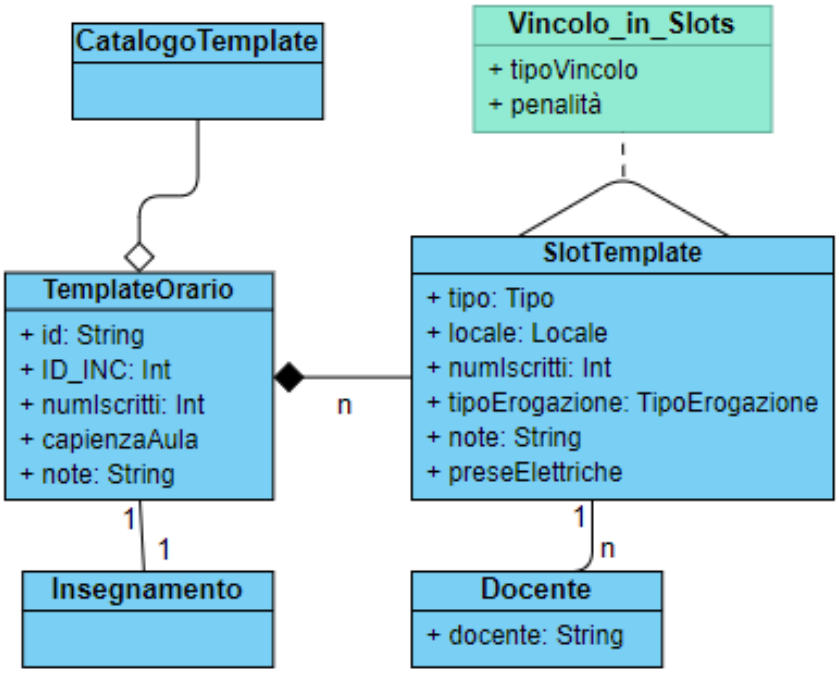


Figura 4.5: Modello UML che descrive il TemplateOrario.

- *id*: costituisce l'identificativo dell'Insegnamento per l'utente (mentre per il software risulta essere *ID_INC*) e corrisponde al titolo dell'Insegnamento (sono ammessi duplicati in quanto lo scopo è di rendere maggiormente "user friendly" l'esperienza d'uso del software.
- *capienzaAula*: i Docenti potranno scegliere arbitrariamente la capienza delle Aule da assegnare ai loro Slot, tuttavia questa scelta è preferibilmente da evitare in quanto sarebbe più corretto affidare la gestione della capienza dell'Aula rischiesta in un dato Slot al software stesso in quanto il software è in grado di dedurre la capienza adatta allo Slot in funzione del *numeroIscritti* all'Insegnamento o allo Slot (se specificato).

L'entità *SlotTemplate* rappresenta l'unita elementare allocabile nell'orario prevedento tra le altre cose:

- *tipo*, *tipoErogazione* e *note*: attributi solamente descrittivi che permettono un'immediata lettura delle soluzioni generate.

- *locale numIscritti*, e *preseElettriche*: determinano la scelta del Locale da assegnare allo Slot.
- *docente*: tenendo traccia dei Docenti associati allo Slot è possibile evitare casi di sovrapposizione che porterebbero uno o più Docenti ad aver contemporaneamente allocate più lezioni.
- *Vincolo_in_Slots*: rispecchia i criteri già introdotti nel modello UML globale.

I tipi associati agli attributi sono riportati qui di seguito:

```
capienzaAula in ['Piccola', 'Media', 'MedioGrande', 'Grande']
tipoLezione in [L, EA, EL, L_EA]
tipoErogazione in ['Presenza', 'Remoto']
locale in ['Aula', 'Aula multimediale', 'Aula attrezzata CA2', 'Aula
    attrezzata CA', 'Aula TableBox', 'Aula WallBox', 'LABINF', 'LAIB',
     'LADISPE', 'LED', 'LED1', 'LED2', 'ACSLAB', 'Aula 5T']
livelloVincolo in [LVH, LV1, LV2, LV3, LV4, LV5, LV_MAX]
tipoVincolo in ['almenoUnGiornoDopo', 'subitoDopo', 'stessoGiorno', '
    giornoDiverso', 'parallelo']
preseElettriche in ['Si', 'No', 'Preferibile']
```

- I tipi di Slot sono riportati in Figura 4.6. Viene estesa l'entità Slot

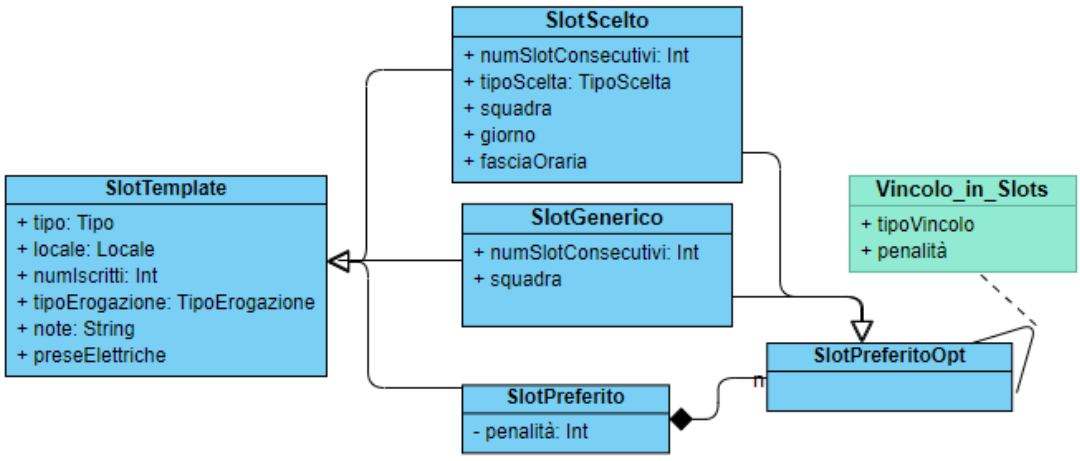


Figura 4.6: Modello UML che descrive i tipi di slot.

osservando che non è possibile un approccio completamente uniforme in quanto alcuni di essi abbiano requisiti molto più stringenti di altri per vari motivi. Si progettano quindi tre entità derivate da *SlotTemplate*:

- *SlotScelto*: consente di raccogliere i requisiti più stringenti in assoluto per uno Slot permettendo l'allocazione di esso in un giorno e/o ad un'ora precisa.
- *SlotGenerico*: rappresenta uno Slot che non ha prerequisiti in termini di giorno e fasciaOraria. L'attributo *squadra* comune ad entrambi i tipi di Slot consente di gestire correttamente lezioni da non allocare all'intera platea di studenti ma solo ad una sua sottoparte permettendo in assenza di altri vincoli un'allocazione eventualmente parallela di più Slot[8].
- *SlotPreferito*: consente ai Docenti di definire opzioni multiple più o meno appetibili a seconda dell'attributo *penalità*. Ognuna di queste opzioni sarà costituita da uno o più Slot dei due tipi elementari precedentemente introdotti.

I tipi associati agli attributi sono presentati di seguito:

```
tipoScelta in ['SlotPreciso', 'NonPrima', 'NonDopo']
squadra in ['Squadra 1', 'Squadra 2', 'Squadra 3', 'Squadra 4']
giorno in [Lun, Mar, Mer, Gio, Ven, Sab, 'Indifferente']
fasciaOraria in [8.30-10.00, 10.00-11.30, 11.30-13.00, 13.00-14.30,
    14.30-16.00, 16.00-17.30, 17.30-19.00, 'Indifferente']
livelloVincolo in [LVH, LV1, LV2, LV3, LV4, LV5]
tipoVincolo in ['almenoUnGiornoDopo', 'subitoDopo', 'stessoGiorno', '
    giornoDiverso', 'parallelo']
```

4.2 Sorgenti di dati

Di seguito vengono introdotte tutte le sorgenti di dati da cui si sono ricavati i dati necessari ad un corretto funzionamento del software.

[8]L'allocazione parallela di Slot non è forzata (esplicabile come vincolo), ma solo consentita in assenza di vincoli quali la presenza di Docenti differenti nei differenti Slot appartenenti a squadre diverse.

4.2.1 Template Orario

Successivamente alla progettazione di alto livello del modello UML del Template Orario è stato implementato un modello di basso livello permettendo una rapida validazione delle differenti descrizioni degl Insegnamenti. Per la rappresentazione dei dati si è scelto il linguaggio XML[9] per la sua struttura versatile e la possibilità di definire uno schema da rispettare rendendo la validazione dell'input immediata e visibile dai vari editor di testo disponibili sul mercato[10]. Di seguito alcune parti del file *schemaUseCase.xsd* contenente lo schema di validazione per i file XML contenenti i Template Orario in input al software.

- Catologo Template:

```
<xs:element name="CatalogoTemplate">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="TemplateOrario" minOccurs="1" maxOccurs="
    unbounded"></xs:element>
      </xs:sequence>
    </xs:complexType>
</xs:element>
```

  Costituisce l'entità radice di un file XML valido. Permette di definire nello stesso file differenti Template Orario riguardanti quindi Insegnamenti diversi, tale scelta è dovuta al fatto che può capitare che in caso di istanza multiple di un Insegnamento, queste abbiano un elevato grado di affinità o che esista un Docente titolare di più di uno di essi; in questi casi la possibilità di avere una visione globale dei dati ne favorisce una corretta lettura e interpretazione.

- Template Orario:

```
<xs:element name="TemplateOrario">
```

[9] eXtensible Markup Language

[10] Nello sviluppo della Tesi è stato usato di VisualCode.

```
    <xs:complexType>
        <xs:sequence>
            <xs:element name="id" minOccurs="1" maxOccurs="1"></xs:element>
            <xs:element name="ID_INC" type="xs:int" minOccurs="1" maxOccurs="1"></xs:element>
            <xs:element ref="NumIscritti" minOccurs="1" maxOccurs="1"></xs:element> <!-- se noto, altrimenti 0, verrà estratto dal GOF -->
            <xs:element ref="NumIscrittiPrimaFrequenza" minOccurs="0" maxOccurs="1"></xs:element>
            <!-- se noto, altrimenti 0, verrà estratto dal GOF -->
            <xs:element ref="CapienzaAula" minOccurs="0" maxOccurs="1"></xs:element> <!-- se il Docente vuole specificare la capienza dell'Aula -->
            <xs:element ref="SlotScelto" minOccurs="0" maxOccurs="unbounded"></xs:element>
            <xs:element ref="SlotGenerico" minOccurs="0" maxOccurs="unbounded"></xs:element>
            <xs:element ref="Vincolo" minOccurs="0" maxOccurs="unbounded"></xs:element>
            <xs:element ref="Operatore" minOccurs="0" maxOccurs="unbounded"></xs:element>
            <xs:element name="Note" type="xs:string" minOccurs="0" maxOccurs="1"></xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Rappresentazione omonima dell'entità nel modello XML; vengono introdotte le cardinalità degli attributi andando ad esplicitare quelli obbligatori, facoltativi o istanziabili con cardinalità multipla[11]. Si evidenzia inoltre la presenza di degli elementi *Vincolo* e *Operatore* non introdotti esplicitamente nel modello UML che si occupano di modellare la relazione *Vincolo_in_Slots* della rappresentazione di alto livello; infine non è presente alcun riferimento al tipo *SlotScelto*[12]. In seguito alla rimozione del tipo *SlotPreferito* risulta essere superflua la classe SlotTemplate del

[11] Rispettivamente: *minOccurs="1"*, *minOccurs="1"* e *maxOccurs="unbounded"*.

[12] Questa scelta implementativa è dettata dall'impossibilità di un'implementazione efficace di tale entità senza andare ad inficiare il modello matematico. Si faccia riferimento al Capitolo 5 per ulteriori dettagli.

modello UML se si procede istanziando all'interno di *TemplateOrario* direttamente elementi di tipo *SlotScelto* e *SlotGenerico*.

- Slot Scelto e Slot Generico:

```
<xs:element name="SlotScelto">
 <xs:complexType>
   <xs:sequence>
     <xs:element ref="Docente" minOccurs="1" maxOccurs="unbounded"></xs:element>
     <xs:element ref="TipoErogazione" minOccurs="0" maxOccurs="1"></xs:element> <!-- default: Presenza-->
     <xs:element ref="NumSlotConsecutivi" minOccurs="1" maxOccurs="1"></xs:element>
     <xs:element ref="Tipo" minOccurs="1" maxOccurs="unbounded"></xs:element>
     <xs:element ref="NumIscritti" minOccurs="0" maxOccurs="1"></xs:element> <!-- Da inserire solo se diverso dal numero di iscritti all'Insegnamento -->
     <xs:element ref="Squadra" minOccurs="0" maxOccurs="1"></xs:element> <!-- default: no Squadra -->
     <xs:element ref="PreseElettriche" minOccurs="0" maxOccurs="1"></xs:element> <!-- default: no prese elettriche necessarie -->
     <xs:element ref="Locale" minOccurs="1" maxOccurs="1"></xs:element> <!-- default: Aula -->
     <xs:element ref="TipoScelta" minOccurs="0" maxOccurs="1"></xs:element> <!-- default: SlotPreciso -->
     <xs:element ref="Giorno"></xs:element> <!-- Indicano il giorno e l'ora in cui allocare lo Slot (in caso di NumSlotConsecutivi > 1 indicano il primo slot della lezione) -->
     <xs:element ref="FasciaOraria"></xs:element>
     <xs:element ref="Nota" minOccurs="0" maxOccurs="1"></xs:element>
   </xs:sequence>
   <xs:attribute name="slotId" type="xs:string" use="required"></xs:attribute>
 </xs:complexType>
</xs:element>

 <xs:element name="SlotGenerico">
 <xs:complexType>
   <xs:sequence>
     <xs:element ref="Docente" minOccurs="1" maxOccurs="unbounded"></xs:element>
```

```
25          <xs:element ref="TipoErogazione" minOccurs="0" maxOccurs="1">
    </xs:element> <!-- default: Presenza-->
26          <xs:element ref="NumSlotConsecutivi" minOccurs="1" maxOccurs=
    "1"></xs:element>
27          <xs:element ref="Tipo" minOccurs="1" maxOccurs="unbounded"></
    xs:element>
28          <xs:element ref="NumIscritti" minOccurs="0" maxOccurs="1"></
    xs:element> <!-- Da inserire solo se diverso dal numero di
    iscritti all'Insegnamento -->
29          <xs:element ref="Squadra" minOccurs="0" maxOccurs="1"></
    xs:element> <!-- default: no Squadra -->
30          <xs:element ref="PreseElettriche" minOccurs="0" maxOccurs="1"
    ></xs:element> <!-- default: no prese elettriche necessarie -->
31          <xs:element ref="Locale" minOccurs="0" maxOccurs="1"></
    xs:element> <!-- default: Aula -->
32          <xs:element ref="Nota" minOccurs="0" maxOccurs="1"></
    xs:element>
33        </xs:sequence>
34        <xs:attribute name="slotId" type="xs:string" use="required"></
    xs:attribute>
35       </xs:complexType>
36    </xs:element>
```

Gli attributi presenti nel modello UML sono stati spostate in entrambe le entità figlie. Vengono prese le seguenti decisioni in fase implementativa circa la cardinalità e l'opzionabilità degli attributi:

- *Docente*: elemento obbligatorio e presente anche in più istanze poichè certi Slot potrebbero essere tenuti da più docenti contemporaneamente o più probabilmente in periodi diversi del semestre.
- *NumSlotConsecutivi*, *Tipo*, *Locale*, *Giorno* e *FasciaOraria* sono elementi obbligatori.
- *TipoErogazione*, *NumIscritti*, *Squadra*, *PreseElettriche*, *TipoScelta*[13] e *Nota*: questi elementi aggiungono solo informazioni supplementari; è previsto un valore di default conservativo[14].

[13]Inizialmente è stata prevista la possibilità di scegliere tra *SlotPreciso*, *NonPrima* e *NonDopo*, successivamente si è preferito implementare un vincolo di questo genere a livello di Docente e/o Insegnamento piuttosto che di singolo Slot

[14]Si usa un approccio "Ask if you need" per allocare solamente le risorse strettamente necessarie, essendo queste in numero limitato.

  - Attributo *slotId*: obbligatorio per permettere di identificare in modo univoco l'elemento Slot tra tutti quelli del Template Orario[15]. La distinzione univoca tra Slot è necessaria per una corretta definizione dei vincoli tra Slot.

- Vincolo

```
1     <xs:element name="Vincolo">
2    <xs:complexType>
3      <xs:all>
4        <xs:element ref="TipoVincolo"></xs:element>
5        <xs:element name="slotId1" type="xs:string" minOccurs="1"
   maxOccurs="1"></xs:element>
6        <xs:element name="slotId2" type="xs:string" minOccurs="1"
   maxOccurs="1"></xs:element>
7        <xs:element ref="Penalita" minOccurs="0" maxOccurs="1"></
   xs:element> <!-- Viene omesso nel caso si tratti di parte di un
   Vincolo esterno -->
8      </xs:all>
9      <xs:attribute name="vincoloId" type="xs:string" use="required">
   </xs:attribute>
10     <xs:attribute name="globalSlot1Id" type="xs:boolean" use="
   optional" default="false"></xs:attribute>
11     <xs:attribute name="globalSlot2Id" type="xs:boolean" use="
   optional" default="false"></xs:attribute>
12    </xs:complexType>
13  </xs:element>
14
```

L'elemento rappresenta un vincolo elementare tra due elementi *SlotGenerico* o *SlotScelto* appartenenti al Template Orario. Gli elementi *TipoVincolo* e *Penalita* sono allineati con la loro progettazione di alto livello nel modello UML. In seguito a richieste particolari emerse in fase avanzata di sviluppo viene introdotta un'evolutiva per consentire la definizione di elementi *Vincolo* che si riferiscono anche a Template Orario differenti tramite la definizione degli attributi opzionali *globalSlot1Id* e *globalSlot2Id* che permettono di collegare il vincolo non a due Slot

[15]L'identificativo è globale ignorando se lo Slot corrente è di tipo *SlotScelto* o di tipo *SlotGenerico*.

del Template Orario corrente, ma a Slot qualsiasi presenti nell'intero dataset[16].

- Operatore

```
      <xs:element name="Operatore">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Vincolo" minOccurs="0" maxOccurs="unbounded"
    ></xs:element>
        <xs:element ref="Operatore" minOccurs="0" maxOccurs="
    unbounded"></xs:element>
        <xs:element ref="Penalita" minOccurs="0" maxOccurs="1"></
    xs:element>
      </xs:sequence>
      <xs:attribute name="id" type="TipoOperatore" use="required"></
    xs:attribute>
    </xs:complexType>
  </xs:element>

    <xs:simpleType name="TipoOperatore">
    <xs:restriction base="xs:string">
      <xs:enumeration value="OR"></xs:enumeration>
      <xs:enumeration value="AND"></xs:enumeration>
      <xs:enumeration value="NOT"></xs:enumeration>
    </xs:restriction>
  </xs:simpleType>

```

L'introduzione di questo nuovo elemento permette di superare le limitazioni presenti con il solo elemento Vincolo consentendo la definizione di vincoli costituiti da una logica booleana più complessa tramite l'annidamento di elementi Vincolo e Operatore all'interno di un Operatore radice[17].

Esempio 9. Viene presentato ora un esempio minimale di Template Orario che rispetta lo schema sinora descritto e che rispecchi fedelmente il caso d'uso seguente:

[16] L'evolutiva è una violazione del modello UML, tuttavia si è adottata per la semplicità di implementazione abbinata al numero limitato di modifiche che richiedeva.

[17] La penalità in questo caso assume valore solamente nell'elemento radice.

1. “Il corso di Informatica presenta due diverse lezioni rispettivamente da 1.5h e 3h e un laboratorio articolato su due turni da 1.5h ciascuno da allocare consecutivamente”.
2. “Le lezioni sono tenute solitamente dal docente Paolo Rossi, anche se saltuariamente la lezione da 1.5h è tenuta dal docente Marco Bianco. I laboratori sono gestiti entrambi da Francesco Neri.”
3. “Il corso prevede 200 iscritti alla prima frequenza e si cerca di ripartire in modo uniforme gli studenti tra i due laboratori”.
4. “Data la situazione incerta data dalla pandemia per una maggiore tutela degli studenti si decide di evitare l’uso dei laboratori universitari ma di utilizzare delle Aule con le prese elettriche in modo da permettere l’uso dei computer personali”.
5. “É stabilito di non avere più di 3 ore di lezione lo stesso giorno e si desidera che i laboratori siano l’ultima lezione erogata nell’arco della settimana al fine di affrontare prima da un punto di vista teorico gli argomenti”.
6. “Si vuole effettuare una lezione il Lunedì”.

Di seguito la descrizione in XML dei requisiti:

```
<TemplateOrario>
<id>Informatica</id>
<ID_INC>11111</ID_INC>
<NumIscritti>200</NumIscritti>
<SlotScelto slotId="slot1"><Docente>Rossi Paolo</Docente><
    NumSlotConsecutivi>2</NumSlotConsecutivi><Tipo>L</Tipo><Locale>Aula</
    Locale><Giorno>Lun</Giorno><FasciaOraria>Indifferente</FasciaOraria></
    SlotScelto>
<SlotGenerico slotId="slot2"><Docente>Rossi Paolo</Docente><Docente>Marco
     Bianco</Docente><NumSlotConsecutivi>1</NumSlotConsecutivi><Tipo>L</
    Tipo><Locale>Aula</Locale></SlotGenerico>
<SlotGenerico slotId="slotLab"><Docente>Neri Francesco</Docente><
    NumSlotConsecutivi>2</NumSlotConsecutivi><Tipo>EA</Tipo><NumIscritti>
    100</NumIscritti><PreseElettriche>Si</PreseElettriche><Locale>Aula
    attrezzata CA2</Locale></SlotGenerico>

<Vincolo vincoloId="vinc1"><Penalita>LVH</Penalita><slotId1>slotLab</
    slotId1><slotId2>slot1</slotId2><TipoVincolo>almenoUnGiornoDopo</
    TipoVincolo></Vincolo>
<Vincolo vincoloId="vinc2"><Penalita>LVH</Penalita><slotId1>slotLab</
    slotId1><slotId2>slot2</slotId2><TipoVincolo>almenoUnGiornoDopo</
    TipoVincolo></Vincolo>
<Vincolo vincoloId="vinc3"><Penalita>LVH</Penalita><slotId1>slot1</
    slotId1><slotId2>slot2</slotId2><TipoVincolo>giornoDiverso</
    TipoVincolo></Vincolo>
</TemplateOrario>
```

4.2.2 Aule disponibili

Un'ulteriore fonte di informazioni essenziale per il software sono i Locali resi disponibili dall'università per allocare gli Slot. Al fine di uniformare le sorgenti di informazioni e di evitare il verificarsi di ambiguità è presente un ulteriore schema per validare i file rappresentanti i Locali allocabili. Di seguito alcuni dei cardini di tale modello:

- Locali Allocati e Gruppo di Locali

```
1     <xs:element name="LocaliAllocati">
2         <xs:complexType>
3             <xs:sequence>
4                 <xs:element ref="GruppoLocali" minOccurs="1"
    maxOccurs="unbounded"></xs:element>
5             </xs:sequence>
6         </xs:complexType>
7     </xs:element>
8
9     <xs:element name="GruppoLocali">
10        <xs:complexType>
11            <xs:sequence>
12                <xs:element ref="Nome"></xs:element>
13                <xs:element ref="Locale"></xs:element>
14                <xs:element ref="Capienza"></xs:element>
15                <xs:element ref="PreseElettriche"></xs:element>
16                <xs:element ref="Quantita"></xs:element>
17                <xs:element ref="Da"></xs:element>
18                <xs:element ref="A"></xs:element>
19            </xs:sequence>
20        </xs:complexType>
21    </xs:element>
22
```

  I Locali Allocati sono un elenco di Gruppi di Locali, in cui ognuno di questi rappresenta la disponibilità di un certo numero di Locali di uno stesso tipo (ed eventualmente di una data capienza) in un arco temporaneo in quanto non è garantito che i Locali liberi possano mantenersi gli stessi per tutta la settimana.

- Locali

```
<xs:element name="Locale">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="Aula multimediale"></xs:enumeration>
        <xs:enumeration value="Aula"></xs:enumeration> <!-- Default value -->
        <xs:enumeration value="Aula attrezzata"></xs:enumeration>
        <xs:enumeration value="Laboratorio"></xs:enumeration> <!-- Tutti i lab vanno bene -->
        <xs:enumeration value="LABINF"></xs:enumeration>
        <xs:enumeration value="LAIB"></xs:enumeration>
        <xs:enumeration value="LADISPE"></xs:enumeration>
        <xs:enumeration value="LED"></xs:enumeration>
        <xs:enumeration value="LED1"></xs:enumeration> <!-- LED1 e LED2 da trattati come il LABINF -->
        <xs:enumeration value="LED2"></xs:enumeration>
        <xs:enumeration value="ACSLAB"></xs:enumeration>
        <xs:enumeration value="Aula 5T"></xs:enumeration>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
```

Una lettura approfondita delle tipologie di locali da selezionare fa saltare all'occhio la mancanza dei valori: *Aula attrezzata CA2*, *Aual attrezzata CA*, *Aula TableBox* e *Aula Wallbox*. Si considerano tutti i casi elencati appartenenti al tipo generico Aula[18], restando fermo il fatto che l'abbinamento di uno Slot in un locale di tipo Aula o derivati possa avvenire solamente se resta rispettato il vincolo sulle presenza di prese elettriche (se richiesto) e se la capienza dell'aula è congrua al numero di studenti o eventualmente superiore[19].

Esempio 10. Di seguito un esempio elementare per definire la seguente disponibilità di Locali:

[18]Si tratta di una scelta implementativa resasi necessaria per raggiungere un compromesso tra precisione del modello e performance del software.

[19]Funzionalità opzionale attivabile in sede di configurazione del software. Fare riferimento al manuale utente.

```xml
<LocaliAllocati xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="../models/schemaAule.xsd">
    <GruppoLocali>
        <Nome>Aule piccole</Nome>
        <Locale>Aula</Locale>
        <Capienza>Piccola</Capienza>
        <PreseElettriche>Si</PreseElettriche>
        <Quantita>2</Quantita>
        <Da>
            <Giorno>Lun</Giorno>
            <FasciaOraria>8.30-10.00</FasciaOraria>
        </Da>
        <A>
            <Giorno>Mar</Giorno>
            <FasciaOraria>17.30-19.00</FasciaOraria>
        </A>
    </GruppoLocali>
    <GruppoLocali>
        <Nome>Aule piccole</Nome>
        <Locale>Aula</Locale>
        <Capienza>Piccola</Capienza>
        <PreseElettriche>Si</PreseElettriche>
        <Quantita>2</Quantita>
        <Da>
            <Giorno>Mer</Giorno>
            <FasciaOraria>8.30-10.00</FasciaOraria>
        </Da>
        <A>
            <Giorno>Ven</Giorno>
            <FasciaOraria>17.30-19.00</FasciaOraria>
        </A>
    </GruppoLocali>
    <GruppoLocali>
        <Nome>LED</Nome>
        <Locale>LED</Locale>
        <Capienza>NonDisponibile</Capienza>
        <Quantita>4</Quantita>
        <Da>
            <Giorno>Lun</Giorno>
            <FasciaOraria>8.30-10.00</FasciaOraria>
        </Da>
        <A>
            <Giorno>Ven</Giorno>
            <FasciaOraria>17.30-19.00</FasciaOraria>
```

```
        </A>
    </GruppoLocali>
</LocaliAllocati>
```

4.2.3 Richieste Docenti

Dai requisiti del software è evidente la varietà di vincoli che i Docenti possono esprimere e di cui è necessario tenere conto per ottenere una soluzione. Si è deciso di rappresentare tutti i vincoli esprimibili dai Docenti sotto forma di file JSON, di cui ora verranno evidenziate le tipologie preferenze espresse:

Esempio 11. Preferenza di giorni e fasce orarie in cui devono svolgersi gli Slot di un Insegnamento (*insegnamenti.json*).

```
{
    "preferenze": {
        "11111": {
            "favorevole": {
                "Lun_16.00-17.30_LV5": "primaDi"
            },
            "contrario": {
                "Lun_8.30-10.00_LV2": "ugualeA",
                "Lun_17.30-19.00_LVH": "ugualeA",
                "Indifferente_16.00-17.30_LV3": "dopoDi"
            }
        }
    }
}
```

Nell'Esempio 11 vengono definiti per ogni Insegnamento un elenco di pareri favorevoli e/o contrari e ognuno di essi viene abbinato ad uno slot temporale, ad un livello di penalità e ad un vincolo di logica temporale[20].

Esempio 12. Preferenza di un Docente a tenere e/o non tenere lezione in determinati giorni e/o fasce orarie (*docenti.json*).

[20]Le logiche temporali ammesse nel modello sono: *primaDi*, *ugualeA* e *dopoDi*.

```
{
    "preferenze": {
        "Francesco Neri": {
            "favorevole": {
                "Lun_16.00-17.30_LV1": "primaDi"
            },
            "contrario": {
                "Lun_8.30-10.00_LV0": "ugualeA",
                "Lun_17.30-19.00_LV0": "ugualeA"
            }
        },
    }
}

```

Nell'Esempio 12 vengono definiti per ogni Docente un elenco di pareri favorevoli e/o contrari e ognuno di essi viene abbinato ad uno slot temporale, ad un livello di penalità e ad un vincolo di logica temporale[21].

Esempio 13. Numero massimo di ore di lezione allocate quotidianamente per un singolo Docente (*docenti.json*).

```
{
    "Docenti": {
        "maxSlotPerDay": {
            "default": 4,
            "Francesco Neri": 5
        },
    }
}
```

Viene definito come vincolo generale un massimo di 4 slot orari[22] come limite di lezioni tenute da un Docente in un dato giorno. Tale limite è tuttavia personalizzabile per il singolo Docente in caso di particolari richieste come mostrato nell'Esempio 13.

Esempio 14. Indicazione su Insegnamenti sovrapponibili anche se tenuti dallo stesso

[21] Le logiche temporali ammesse nel modello sono: *primaDi*, *ugualeA* e *dopoDi*.

[22] 6 ore.

Docente[23].

```
{
    "Docenti": {
        "insegnamentiSovrapponibili":{
            "Francesco Neri":[254394,254395,254396]
        },
    }
}

```

Esempio 15. Preferenza di un Docente ad utilizzare l'orario dell'anno precedente (*docenti.json*).

```
{
    "Docenti": {
        "preferenzaAnnoPrecedente": {
            "Francesco Neri": "prova1_LV2"
        },
    }
}

```

Nell'esempio 15 tutti gli Slot associati al Docente indicato nel piano di allocazione specificato ("prova1" in questo caso) saranno allocati tenendo conto dell'allocazione precedente e andando incontro alla penalità specificata in caso non vengano rispettati.

Esempio 16. Distanza minima/massima tra Slot tenuti dallo stesso Docente se allocati lo stesso giorno ed eventuale penalità in caso questi Slot appartengano ad Insegnamenti differenti (*docenti.json*).

```
{
    "Docenti": {
```

[23]Si tratta di un caso particolare in cui un Docente tiene lezioni in Insegnamenti differenti in periodi diversi del semestre e che quindi non devono risultare in conflitto tra loro.

```
        "distanzaMinima": {
            "Francesco Neri": "1_LV1"
        },
        "distanzaMassima": {
            "Francesco Neri": "1_LV2"
        },
        "insegnamentiDiversiSameDay": {
            "Francesco Neri": "LV1"
        }
    }
}
```

Nell'Esempio 16 sono presenti preferenze riguardanti la distanza minima e massima il numero intero concatenato al livello di penalità con un "_" corrisponde al numero di slot orari di distanza.

Tutte le tipologie di vincolo illustrate in questa sezione di default non sono attive, ma devono essere esplicitate nei file JSON indicati dagli esempi.

4.2.4 Insegnamenti in un Orientamento

La descrizione di tutti gli Orientamenti e i CDL in cui compare un Insegnamento è un'ulteriore fonte di dati per il software. Essi vengono salvati all'interno di un database di appoggio seguendo la progettazione UML di Figura 4.1.

Il modello logico dell'Orientamento di Figura 4.7 prevede anche l'adozione di un nuovo attributo *alfabetica* poichè è necessario non generare vincoli tra Insegnamenti appartenenti ad alfabetiche differenti[24].

4.3 Formato soluzione

Il tool durante la sua esecuzione salva le soluzioni all'interno del database secondo la progettazione UML di Figura 4.3. Il modello logico dello Slot rispecchia la progettazione di alto livello a meno di un ulteriore attributo note

[24]Fissato un Orientamento in cui sono presenti due istanze di Insegnamenti distinguibili tra loro per una diversa valorizzazione dell'attributo alfabetica, sarebbe scorretto impedire in assenza di vincoli di natura differente l'allocazione parallela di Insegnamenti anche obbligatori per tale Orientamento se appartenenti a differenti alfabetiche.

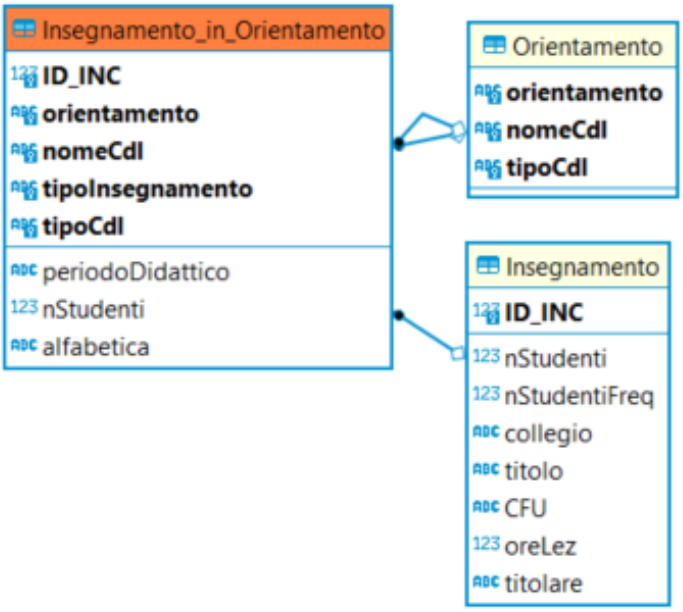


Figura 4.7: Modello logico Insegnamento in Orientamento.

inserito per consentire di aggiungere del testo che potrebbe rivelarsi utile in una fase successiva all'export della soluzione nel caso di necessità particolari per lo Slot[25] come mostrato nel modello logico dello Slot in Figura 4.8.

[25]Tale campo viene esportato in output come spiegato nella Sezione 7.3.4.

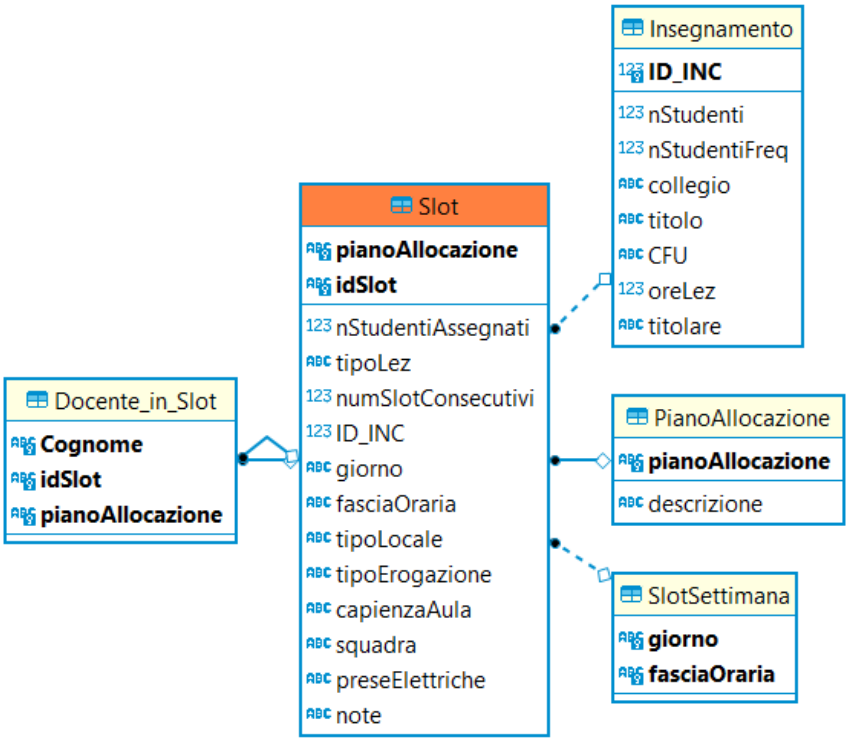


Figura 4.8: Modello logico Slot.

Capitolo 5

Modellazione

In seguito all'analisi dei requisiti vengo estrapolati due modelli differenti candidati a modellare il problema in esame: il *Modello a Matrice Multidimensionale* e il *Modello a Slot*.
La scelta definitiva di adottare un modello a scapito di un altro è dovuta sia alla possibilità di modellare un numero superiore di vincoli che a performance di calcolo più efficienti.

5.1 Modello a Matrice Multidimensionale v1

5.1.1 Modellazione formale

Variabili:

- *Insegnamenti* indice i $\in [1, \text{nIns}] = \text{I}$, I è l'insieme di tutti gli Insegnamenti del modello[1].
- *Docenti* indice d $\in [1, \text{nDoc}] = \text{D}$, D è l'insieme di tutti i Docenti[2].
- *Giorno* indice g $\in [1, 6] = \text{G}$, G è l'insieme delle fasce orarie allocabili dedotto dai requisiti.

[1]Comprende sia gli Insegnamenti da allocare che quelli già allocati e assunti come vincoli per il modello

[2]I Docenti presi in considerazione sono solo quelli che si riferiscono ad almeno uno Slot.

- *Fascia oraria* indice f $\in [1,7]$ = F, F è l'insieme delle fasce orarie allocabili dedotto dai requisiti.
- Insieme di variabili Booleane $\text{mat}_{\text{i,d,g,f}} \in \{0,1\}$ destinate all'allocazione con cardinalità $|I|\times|D|\times|G|\times|F|$[3]:

$$\text{mat\_i,d,g,f} = \begin{cases} 1, \text{ slot dell'Insegnamento } i \text{ allocato per il Docente } d \\ 0, \text{ slot dell'Insegnamento } i \text{ non allocato per il Docente } d^3 \end{cases}$$

Funzioni:

- Insegnamenti di un Docente: una funzione *assignDocente(d)* stabilisce gli Insegnamenti assegnati ad un Docente[4]:

$$\begin{aligned} &\textit{assignDocente} : D \to 2^I \\ &d \mapsto I_d \subseteq I \end{aligned} \tag{5.1}$$

- Indisponibilità di un Docente: una funzione *indispDocente(d)* stabilisce gli slot orari in cui un Docente è impossibilitato a svolgere lezione:

$$\begin{aligned} &\textit{indispDocente} : D \to 2^{G\times F} \\ &d \mapsto \{(\text{g},\text{f})\} \subseteq G \times F \end{aligned} \tag{5.2}$$

- Preferenza di un Docente: una funzione *prefDocente(d)* stabilisce gli slot orari in cui un Docente vuole fare lezione:

$$\begin{aligned} &\textit{prefDocente} : D \to 2^{G\times F} \\ &d \mapsto \{(\text{g},\text{f})\} \subseteq G \times F \end{aligned} \tag{5.3}$$

- Numero di Slot di un Insegnamento: una funzione *numSlot(i)* stabilisce il numero di slot orari da 1.5h settimanali per un Insegnamento:

$$\begin{aligned} &\textit{numSlot} : I \to \mathbb{N} \\ &i \mapsto n_i \in \mathbb{N} \end{aligned} \tag{5.4}$$

[3]L'ordine di grandezza del problema oggetto di Tesi risulta essere approssimativamente: $200 * 300 * 6 * 7 = 2.52 * 10^6$.

[4]Semplificazione nei vincoli in cui si considera che tutti gli Slot di un Insegnamento siano tenuti dagli stessi Docenti

- Max slot orari Docente giornalieri: una funzione *maxSlotDocente(d)* restituisce il numero massimo di slot orari di 1.5h allocabili per un singolo Docente:

$$\begin{aligned} &\textit{maxSlotDocente} : D \to \mathbb{N} \\ &d \mapsto d_i \in \mathbb{N} \end{aligned} \tag{5.5}$$

- Slot di un Insegnamento: una funzione *slotInsegnamento(i)* stabilisce gli slot orari allocati per un Insegnamento:

$$\begin{aligned} &\textit{slotInsegnamento} : I \to 2^{G \times F} \\ &i \mapsto \{(g, f)\} \subseteq G \times F \end{aligned} \tag{5.6}$$

- Slot scelto per un Insegnamento: una funzione *slotScelti(i)* ritorna gli slot orari in cui un Insegnamento deve essere necessariamente tenuto[5]:

$$\begin{aligned} &\textit{slotScelti} : I \to 2^{G \times F} \\ &i \mapsto \{(g, f)\} \subseteq \textit{slotInsegnamento(i)} \end{aligned} \tag{5.7}$$

- Docenti di un Insegnamento: una funzione *docentiIns(i)* ritorna i Docenti associati ad un Insegnamento:

$$\begin{aligned} &\textit{docentiIns} : I \to 2^{D} \\ &i \mapsto D_i \subseteq D \end{aligned} \tag{5.8}$$

Vincoli:

- Non è possibile assegnare lezioni in slot orari dove i Docenti hanno indicato incompatibilità:

$$d \in D, \forall\, (g, f) \in \textit{indispDocente(d)}, \forall i \in I : mat_{i,d,g,f} = 0 \tag{5.9}$$

- Non viene assegnato ad un Docente uno slot orario riferito ad un Insegnamento in cui lui non tiene lezioni:

$$d \in D, \forall i \notin \textit{assingDocente(d)}, \forall g \in G, \forall f \in F : mat_{i,d,g,f} = 0 \tag{5.10}$$

[5]Non necessariamente tutti gli Slot di un Insegnamento sono preallocati.

- Non vengono allocati sovrapposti due Slot appartenenti allo stesso Docente:

$$d \in D, Idoc := assignDocente(d), \forall g \in G, \forall f \in F, \sum_{i \in Idoc} mat_{i,d,g,f} <= 1 \tag{5.11}$$

- Viene allocato uno Slot negli slot orari in cui il Docente ha espresso preferenza:

$$\begin{gathered} d \in D, Idoc := assignDocente(d), \\ \forall\, (g, f) \in prefDocente(d), \sum_{i \in Idoc} mat_{i,d,g,f} = 1 \end{gathered} \tag{5.12}$$

- Limite massimo di slot orari allocati per Docente:

$$\begin{gathered} d \in D, Idoc := assignDocente(d), \forall g \in G, \\ \sum_{i \in Idoc, f \in F} mat_{i,d,g,f} <= maxSlotDocente(d) \end{gathered} \tag{5.13}$$

- Gli Slot Scelti vanno preallocati:

$$\begin{gathered} i \in I, \forall\, (g, f) \in slotScelti(i), \\ \forall\ d \in docentiIns(i),\ mat_{i,d,g,f} = 1 \end{gathered} \tag{5.14}$$

- Si alloca esattamente il numero di slot orari richiesto per un Insegnamento:

$$\begin{gathered} i \in I, d \in docentiIns(i), \\ \sum_{g \in G, f \in F} mat_{i,d,g,f} = numSlot(i) \end{gathered} \tag{5.15}$$

- Si allocano tutti i Docenti relativi all'Insegnamento qualora venga allocato uno Slot per tale Insegnamento:

$$\begin{gathered} i \in I, g \in G, f \in F, d \in docentiIns(i), \\ mat_{i,d,g,f} = 1 \Rightarrow mat_{i,d^*,g,f} = 1\ \forall d^* \in docentiIns(i) \end{gathered} \tag{5.16}$$

- Slot appartenenti allo stesso Insegnamento contigui se allocati lo stesso giorno[6]:

$$i \in I, d \in docentiIns(i), g \in G, f1 \in F,$$
$$\sum_{0 \leq f < f1} mat_{i,d,g,f} = \sum_{0 \leq f \leq f1} mat_{i,d,g,f} \Rightarrow \sum_{f \in F} mat_{i,d,g,f} = \sum_{0 \leq f < f1} mat_{i,d,g,f} \tag{5.17}$$

Che può essere tradotta in:

$$i \in I, d \in docentiIns(i), g \in G,$$
$$\forall f1,\ 0 \leq f1 < F-1,\ \text{se: } mat_{i,d,g,f1} = 1 \wedge mat_{i,d,g,f1+1} = 0 \Rightarrow \sum_{0 \leq f2 \leq f1} mat_{i,d,g,f2} = \sum_{f \in F} mat_{i,d,g,f} \tag{5.18}$$

5.1.2 Modellazione con equazioni

Per la modellazione di basso livello si utilizza il software Z3 ed in particolare le sue API per Python. Di seguito vengono riportare le equazioni corrispondenti ai vincoli precedentemente discussi, per scelta stilistica si decide di collassare le due dimensioni fasciaOraria F e giorno G in un unica chiamata S avente cardinalità $6 \times 7 = 42$.

Matrice booleana $mat_{i,d,g,f} \rightarrow mat_{i,d,s}$, $s \in S$ è l'insieme di tutti gli slot orari.

```
# Insegnamenti X Docenti X Slot
mat = [[[z3.Bool("ins_%s doc_%s slot_%s" % (insegnamenti[i], docenti[d],
    slot[s]))
             for s in range(len(slot))]
             for d in range(len(docenti))]
             for i in range(len(insegnamenti))]

I = len(insegnamenti)
D = len(docenti)
G = 6
F = 7
```

[6]Se per un dato Insegnamento un certo slot orario è allocato è il successivo no, allora non potrò avere slot successivi allocati nello stesso giorno.

```
S = G*F
eqs = list()
```

Equazioni:

- Non è possibile assegnare lezioni in slot orari dove i Docenti hanno indicato incompatibilità:

```
for d in range(D):
    for s in range(S):
        if s in indispDocente(d):
            eqs.append(mat[i][d][s] == False)
```

- Non viene assegnato ad un Docente uno slot orario riferito ad un Insegnamento in cui lui non tiene lezioni:

```
for i in range(I):
    for d in range(D):
        if d not in assignDocente(d):
            for s range(S):
                eqs.append(mat[i][d][s] == False)
```

- Non vengono allocati sovrapposti due Slot appartenenti allo stesso Docente:

```
for d in range(D):
    for s in range(S):
        eqs.append(Sum([If(mat[i][d][s], 1, 0) for i in range(I)]) <=
    1))
```

- Viene allocato uno Slot negli slot orari in cui il Docente ha espresso preferenza:

```
for d in range(D):
```

```
    for s in prefDocente(d):
        eqs.append(Sum([If(mat[i][d][s], 1, 0) for i in assignDocente
    (d)]) == 1)
```

- Limite massimo di slot orari allocati per Docente:

```
for i in range(I):
    for d in range(D):
        for g in range(G):
            eqs.append(Sum([If(mat[i][d][g*F+f], 1, 0) for f in range
    (F)]) <= maxSlotDocente(d))
```

- Gli Slot Scelti vanno preallocati:

```
for i in range(I):
    for s in slotScelti(i):
        for d in docentiIns(i):
            eqs.append(mat[i][d][s] == True)
```

- Si alloca esattamente il numero di slot orari richiesto per un Insegnamento:

```
for i in range(I):
    d = docentiIns(i)[0]
    eqs.append(Sum([If(mat[i][d][s], 1, 0) for s in range(S)]) ==
    numSlot(i))
```

- Si allocano tutti i Docenti relativi all'Insegnamento qualora venga allocato uno Slot per tale Insegnamento:

```
for i in range(I):
    for s in range(S):
        d = docentiIns(i)[0]
        for d_ in docentiIns(i)[1:]:
```

```
5               eqs.append(Implies(mat[i][d][s], mat[i][d_][s] == True))
```

- Slot appartenenti allo stesso Insegnamento contigui se allocati lo stesso giorno:

```
1 for i in range(I):
2     d = docentiIns(i)[0]
3     for g in range(G):
4         for f in range(F-1):
5             eq = Implies(And(mat[i][d][g*F+f], Not(mat[i][d][g*F+f
  +1])),
6                          Sum([If(mat[i][d][g*F+f1], 1, 0) for f1 in range(
  f+1)]) == Sum([If(mat[i][d][g*F+f2], 1, 0) for f2 in range(F)]))
7             eqs.append(eq)
```

5.1.3 Conclusioni

Questo modello presenta numerosi punti critici tra cui:

- Non tiene conto della dimensione Aule.
- Suppone che tutti gli Slot di un Insegnamento siano uguali tra loro e quindi riferiti agli stessi Docenti.
- Non si tiene conto di nessun vincolo tra Slot dello stesso Insegnamento.
- Non si tiene conto di nessuna preferenza/penalità[7].
- Non vengono presi in considerazione gli Insegnamenti che non possono essere sovrapposti tra loro a causa della loro appartenenza allo stesso Orientamento.

Si rende quindi necessaria la modellazione di una versione più precisa partendo da quella presa in esame ora.

[7]Si ignorano tutti i soft constraint.

5.2 Modello a Matrice Multidimensionale v2

5.2.1 Modellazione formale

Variabili (oltre a quelle precedentemente definite):

- *Slot* indice s $\in [1, \text{nSlotIns}] = S_i$, sono gli Slot dell'Insegnamento I.
- *Aule* indice a $\in [1, \text{nAule}] = A$, è l'insieme delle Aule allocabili.
- Ridefinizione dell'insieme di variabili Booleane $\text{mat}_{s,d,a,g,f} \in \{0, 1\}$ con cardinalità $|S| \times |D| \times |A| \times |G| \times |F|$[8]:

$$\text{mat\_i,d,g,f} = \begin{cases} 1, \text{ slot allocato per il Docente } dnell'Aulaa \\ 0, \text{ altrimenti.} \end{cases}$$

5.2.2 Conclusioni

Prima di passare a definire i vincoli del nuovo modello si preferisce verificare le performance del software Z3 dato l'aumento di 2 ordini di grandezza delle dimensioni del problema.

```
# Slot_Insegnamenti X Docenti X Aule X Slot_Orari
mat = [[[z3.Bool("slotIns_%s doc_%s aula_%s slotOrario_%s" % (slotIns[si],
    docenti[d], aule[a], slotOrario[so]))
            for so in range(len(slotOrario))]
            for a in range(len(aule))]
            for d in range(len(docenti))]
            for si in range(len(slotIns))]

I = len(slotIns)
D = len(docenti)
A = len(aule)
G = 6
F = 7
S = G*F
```

[8] L'ordine di grandezza risultante corrisponde a $600 * 300 * 60 * 6 * 7 \cong 4.53 * 10^8$.

```
14 eqs = list()
```

L'allocazione di tale matrice richiede troppe risorse[9], ragione per cui si è deciso di utilizzare un modello completamente diverso.

5.3 Modello a Slot

5.3.1 Modellazione formale

Variabili:

- Slot indice $s \in [1, \text{nSlots}] = S$, S è l'insieme di tutti gli Slot del modello.
- Insegnamenti indice $i \in [1, \text{nIns}] = I$, I è l'insieme di tutti gli Insegnamenti del modello[10].
- Orientamenti indice $o \in [1, \text{nOrient}] = O$, O è l'insieme di tutti gli Orientamenti del modello.
- Docenti indice $d \in [1, \text{nDoc}] = D$, D è l'insieme di tutti i Docenti[11].
- $\text{N\_GIORNI} = 6, \text{N\_FASCE\_ORARIE} = 7$.
- Giorno indice $g \in [0, \text{N\_GIORNI}] = G$, G è l'insieme dei giorni.
- Giorno Slot $X\_d_s \in [0, \text{N\_GIORNI}]$.
- Fascia oraria indice $f \in [0, \text{N\_FASCE\_ORARIE}] = F$, F è l'insieme delle fasce orarie.
- Fascia oraria Slot $X\_h_s \in [0, \text{N\_FASCE\_ORARIE}]$.

[9] Da prove sperimentali è emerso come 64 GB di memoria non risultino sufficienti ad allocare il problema, non si hanno informazioni a sufficienza per stimare con attendibilità le risorse effettivamente richieste.

[10] Comprende sia gli Insegnamenti da allocare che quelli già allocati e assunti come vincoli per il modello.

[11] I Docenti presi in considerazione sono solo quelli che si riferiscono ad almeno uno Slot.

- Penalità Insegnamenti indice $p \in [1, \text{nPenalita}] = P$, P è l'insieme di tutte le penalità riguardanti la sovrapposizione di Insegnamenti dello stesso Orientamento e per vincoli riguardanti un Insegnamento nel modello.

- Penalità per sovrapposizione Insegnamenti dello stesso Orientamento e per vincoli riguardanti un Insegnamento $X\_penalties_p \in [0, .., 7]$.

- Penalità studenti indice $pStud \in [1, \text{nPenalitaStudenti}] = \text{P\_STUD}$, P_STUD è l'insieme di tutte le penalità riguardanti gli studenti nel modello.

- Penalità studenti $X\_penaltiesStud_{pStud} \in [-7, ..0, .., 7]$.

- Penalità per preferenze/indisponibilità Docenti indice $pDoc \in [1, \text{nPenalitaDocenti}] = P\_DOC$, P_DOC è l'insieme di tutte le penalità riguardanti i Docenti nel modello.

- Penalità Docenti $X\_penaltiesDoc_{pDoc} \in [-7, .., 0, .., 7]$.

- Funzione obiettivo[12]:

$$\begin{aligned} min \Bigg(& \sum_{p \in P} X\_penalties_p \\ & + \sum_{pStud \in P\_STUD} X\_penaltiesStud_{pStud} \\ & + \sum_{pDoc \in P\_DOC} X\_penaltiesDoc_{pDoc} \Bigg) \end{aligned} \tag{5.19}$$

- Slot orari del sabato: *SlotSabato()*.

- Slot orari allocati per ogni Anno Accademico per ogni Orientamento indice $a, o, a \in [1, 3], o \in [1, \text{nOrient}] = \text{X\_ORIENT}$ è l'insieme di tutti gli slot orari allocati per ogni Anno Accademico dei diversi Orientamenti.

[12]Il software è configurabile al fine di scegliere come funzione obiettivo anche un sottoinsieme dei termini di somma presenti.

- Slot orari allocati per ogni Anno Accademico per ogni Orientamento[13] $X\_orient_{o,a,g,f} \in [0,1]$.

Funzioni:

- Numero di slot orari di uno Slot: una funzione *numSlot(s)* ritorna il numero di slot orari da 1.5h consecutivi dello Slot:

$$\begin{aligned} &numSlot : S \to \mathbb{N} \\ &s \mapsto n_s \in \mathbb{N} \end{aligned} \tag{5.20}$$

- Tipologia di lezione di uno Slot: una funzione *tipoLez(s)* ritorna il TipoLezione associato allo Slot:

$$\begin{aligned} &TipoLezione = \{\mathrm{L}, \mathrm{EL}, \mathrm{EA}, \mathrm{L\_EA}\} \\ &tipoLez : S \to TipoLezione \\ &s \mapsto tipoLezione_s \in TipoLezione \end{aligned} \tag{5.21}$$

- Docenti associati ad uno Slot: una funzione *docentiSlot(s)* ritorna i Docenti associati ad uno Slot:

$$\begin{aligned} &docentiSlot : S \to 2^D \\ &s \mapsto D_s \subseteq D \end{aligned} \tag{5.22}$$

- Max slot orari Docente giornalieri: una funzione *maxSlotDocente(d)* stabilisce il numero massimo di slot orari di 1.5h allocabili per un singolo Docente:

$$\begin{aligned} &maxSlotDocente : D \to \mathbb{N} \\ &d \mapsto i_d \in \mathbb{N} \end{aligned} \tag{5.23}$$

- Slot di un Insegnamento: una funzione *slotInsegnamento(i)* stabilisce gli slot orari allocati per un Insegnamento:

$$\begin{aligned} &slotInsegnamento : I \to 2^S \\ &i \mapsto \{S_i\} \subseteq S \end{aligned} \tag{5.24}$$

[13] Vengono raggruppati insieme i PeriodiDidattici dello stesso anno accademico.

- Slot scelto per un Insegnamento: una funzione *slotScelti(i)* ritorna gli Slot preallocati di un Insegnamento [14]:

$$\begin{aligned} &slotScelti : I \rightarrow 2^S \\ &i \mapsto \{S_i\} \subseteq slotInsegnamento(i) \end{aligned} \tag{5.25}$$

- Giorno e Fascia oraria di uno SlotScelto: una funzione *detailsSlotScelto(s)* ritorna Giorno e Fascia oraria di uno SlotScelto:

$$\begin{aligned} &detailsSlotScelto : S \rightarrow 2^{G \times F} \\ &s \mapsto \{(g, f)\} \subseteq G \times F \end{aligned} \tag{5.26}$$

- Slot tenuti da un Docente: una funzione *slotDocente(d)* restituisce gli Slot allocati per un dato Docente:

$$\begin{aligned} &slotDocente : D \rightarrow 2^S \\ &d \mapsto \{S_d\} \subseteq S \end{aligned} \tag{5.27}$$

- Insegnamenti in un Orientamento: una funzione *insOrientamento(o)* restituisce gli Insegnamenti per un dato Orientamento:

$$\begin{aligned} &insOrientamento : O \rightarrow 2^I \\ &o \mapsto \{I_o\} \subseteq I \end{aligned} \tag{5.28}$$

- Squadra di uno Slot (se presente): una funzione *squadraSlot(s)* ritorna la squadra associata allo Slot (se presente):

$$\begin{aligned} &squadraSlot : S \rightarrow Squadra \\ &s \mapsto squadra_s \in Squadra \end{aligned} \tag{5.29}$$

$$\begin{aligned} Squadra = \{&\text{NoSquadra}, \text{Squadra\_1}, \text{Squadra\_2}, \\ &\text{Squadra\_3}, \text{Squadra\_4}\} \end{aligned}$$

- TipoSlot: una funzione *tipoSlot(s)* ritorna il tipo di Slot:

$$\begin{aligned} &tipoSlot : S \rightarrow TipoSlot \\ &s \mapsto tipoSlot_s \in TipoSlot \end{aligned} \tag{5.30}$$

$$\begin{aligned} TipoSlot = \{&\text{SlotScelto}, \text{SlotGenerico}, \text{SlotScelto\_giorno}, \\ &\text{SlotScelto\_fasciaOraria}\} \end{aligned}$$

[14] Non necessariamente tutti gli Slot di un Insegnamento sono preallocati.

- PeriodoDidattico di un Insegnamento all'interno di un Orientamento: una funzione *periodoDidattico(i, o)* ritorna il periodoDidattico:

$$\begin{aligned} &periodoDidattico : (I, O) \rightarrow PeriodoDidattico \\ &i, o \mapsto periodoDidattico_{i,o} \in PeriodoDidattico \end{aligned}$$

$$\begin{aligned} PeriodoDidattico = \{&\text{PrimoAnno\_PrimoSemestre}, \\ &\text{PrimoAnno\_SecondoSemestre}, \text{SecondoAnno\_PrimoSemestre}, \\ &\text{SecondoAnno\_SecondoSemestre}, \text{TerzoAnno\_PrimoSemestre}, \\ &\text{TerzoAnno\_SecondoSemestre}\} \end{aligned} \tag{5.31}$$

- Alfabetica di un Insegnamento all'interno di un Orientamento: una funzione *alfabetica(i, o)* ritorna l'alfabetica:

$$\begin{aligned} &alfabetica : (I, O) \rightarrow Alfabetica \\ &i, o \mapsto alfabeticai, o \in Alfabetica \end{aligned} \tag{5.32}$$

$$\begin{aligned} Alfabetica = \{&\text{NoAlfabetica}, \text{PrimaAlfabetica}, \\ &\text{SecondaAlfabetica}, \text{TerzaAlfabetica}\} \end{aligned}$$

- Penalita sovrapposizione: una funzione *penalitaSovr(i1,i2,o)* ritorna il livello di penalità da aggiungere al modello in caso di sovrapposizione di due Insegnamenti i1 e i2 per l'Orientamento o[15]:

$$\begin{aligned} &penalitaSovr : (I^2, O) \rightarrow LV\_Penalties \\ &i1, i2, o \mapsto penalita_{i1,i2,o} \in LV\_Penalties \end{aligned} \tag{5.33}$$

$$\begin{aligned} LV\_Penalties = \{&\text{LV\_H}, \text{LV\_0}, \text{LV\_1}, \text{LV\_2}, \text{LV\_3}, \\ &\text{LV\_4}, \text{LV\_5}, \text{LV\_6}, \text{LV\_MAX}\} \end{aligned}$$

- Insegnamenti di un Orientamento in un dato Anno Accademico: una funzione *insInAnnoOrientamento(o,a)* ritorna gli Insegnamenti di un

[15]Dato che un Insegnamento può essere parte di più Orientamenti è possibile che in differenti Orientamenti la penalità del sovrapporsi degli stessi Insegnamenti assuma un valore differente.

Orientamento in un dato Anno Accademico:

$$\begin{aligned} &\textit{insInAnnoOrientamento} : (O, A) \rightarrow 2^I \\ &o, a \mapsto \{I_{o,a}\} \subseteq \textit{insOrientamento(O)} \\ \\ &A = \{1, 2, 3\} \end{aligned} \tag{5.34}$$

- Insegnamenti di tipo Obbligatorio, Obbligatorio a scelta e Suggerito un Orientamento in un dato Anno Accademico: una funzione *kernelInsInAnnoOrientamento(o,a)* ritorna gli Insegnamenti di un Orientamento in un dato Anno Accademico:

$$\begin{aligned} &\textit{kernelInsInAnnoOrientamento} : (O, A) \rightarrow 2^I \\ &o, a \mapsto \{I_{o,a}\} \subseteq \textit{insOrientamento(O)} \\ \\ &A = \{1, 2, 3\} \end{aligned} \tag{5.35}$$

- Numero di Locali disponibili per un dato tipo e una data capienza: una funzione *numLocaliDisponibili(g,f,l,c)* ritorna il numero di Locali di tale tipo e capienza[16] disponinili in uno slot orario[17]:

$$\begin{aligned} &\textit{numLocaliDisponibili} : (G, F, L, C) \rightarrow \mathbb{N} \\ &g, f, l, c \mapsto \mathbb{N} \\ \\ L = \{&\text{Aula}, \text{LABINF}, \text{LAIB}, \text{LADISPE}, \text{LED}, \text{LED1}, \\ &\text{LED2}, \text{ACSLAB}, \text{Aula5T}, \text{Aula\_CA2}, \\ &\text{Aula\_CA}, \text{Aula\_TB}, \text{Aula\_WB}\} \\ C = \{&\text{Piccola}, \text{Media}, \text{MedioGrande}, \text{Grande}\} \end{aligned} \tag{5.36}$$

- Tipo di Locale di uno Slot: una funzione *tipoLocale(s)* restituisce il tipoLocale di uno Slot:

$$\begin{aligned} &\textit{tipoLocale} : (S) \rightarrow L \\ &s \mapsto \textit{tipoLocale} \in L \end{aligned} \tag{5.37}$$

[16] Se valorizzabile per il tipo di Locale corrente.

[17] Esiste una funzione analoga che considera solo i Locali dotati di prese elettriche.

- Capienza del Locale associato ad uno Slot: una funzione *capienzaLocale(s)* restituisce la capienza del locale associato ad uno Slot:

$$\begin{aligned} &capienzaLocale : (S) \rightarrow C \\ &s \mapsto capienza \in C \end{aligned} \tag{5.38}$$

Vincoli:

- Gli Slot devono essere assegnati in fasce orarie e gioni validi:

$$s \in S,\ 0 \leq X\_d_s \leq |G| \ \wedge \ 0 \leq X\_h_s \leq |F| - numSlot(s) \tag{5.39}$$

- Al Sabato sono disponibili meno fasce orarie:

$$s \in S, se : X\_d_s = |G| - 1 \Rightarrow X\_h_s + numSlot(s) \leq SlotSabato() \tag{5.40}$$

- Slot appartenenti allo stesso Insegnamento se allocati lo stesso giorno, tenuti da uno o più Docenti in comune e non Esercitazioni di Laboratorio devono essere consecutivi:

$$\begin{gathered} i \in I, \forall s1, s2 \in slotInsegnamento(i), s1 \neq s2, \\ se : docentiSlot(s1) \cap docentiSlot(s2) \neq \emptyset \ \wedge \\ (tipoLez(s1) \neq EL \vee tipoLez(s2) \neq EL) \wedge \\ X\_d_s1 = X\_d_s2 \Rightarrow \\ (X\_h_{s1} = X\_h_{s2} + numSlot(s2)) \vee (X\_h_{s2} = X\_h_{s1} + numSlot(s1)) \end{gathered} \tag{5.41}$$

- Docente allocato in un solo slot orario alla volta:

$$\begin{gathered} d \in D, \forall s1, s2 \in slotDocente(d), s1 \neq s2, \\ se : X\_d_{s1} = X\_d_{s2} \Rightarrow \\ (X\_h_{s1} \geq X\_h_{s2} + numSlot(s2)) \vee (X\_h_{s2} \geq X\_h_{s1} + numSlot(s1)) \end{gathered} \tag{5.42}$$

- Se uno Slot di un Insegnamento è termina alle 19.00 di un dato giorno, il giorno successivo alle 8.30 non potrà esserci un altro Slot dello stesso

Insegnamento a meno di esercitazioni di laboratorio o di lezioni suddivise per squadre[18]:

$$\begin{gathered} i \in I, \forall s1, s2 \in \textit{slotInsegnamento(i)}, s1 \neq s2, \\ se : \textit{tipoLez(s1)} \neq EL \ \wedge \textit{tipoLez(s2)} \neq EL \ \wedge \\ \textit{squadraSlot(s1)} \neq NoSquadra \ \wedge \textit{squadraSlot(s2)} \neq NoSquadra \ \wedge \\ X\_d_{s1} + 1 = X\_d_{s2} \wedge X\_h_{s2} + \textit{numSlot(s2)} = 7 \\ \Rightarrow X\_h_{s1} > 0 \end{gathered} \tag{5.43}$$

- Slot di un Insegnamento non sovrapposti se non si tratta di Slot relativi a squadre differenti[19]:

$$\begin{gathered} i \in I, \forall s1, s2 \in \textit{slotInsegnamento(i)}, s1 \neq s2, \\ se : \textit{docentiSlot(s1)} \cap \textit{docentiSlot(s2)} = \emptyset \ \wedge \\ \textit{squadraSlot(s1)} \neq NoSquadra \wedge \textit{squadraSlot(s2)} \neq NoSquadra \ \wedge \\ \textit{squadraSlot(s1)} \neq \textit{squadraSlot(s2)} \wedge X\_d_s1 = X\_d_s2 \Rightarrow \\ (X\_h_{s1} \geq X\_h_{s2} + \textit{numSlot(s2)} \vee X\_h_{s2} \geq X\_h_{s1} + \textit{numSlot(s1)}) \end{gathered} \tag{5.44}$$

- Slot scelti di un Insegnamento:

$$\begin{gathered} i \in I, \forall s \in \textit{slotScelti(i)} \Rightarrow \\ X\_d_s = \textit{detailsSlotScelto(s).g} \wedge X\_h_s = \textit{detailsSlotScelto(s).f} \end{gathered} \tag{5.45}$$

- Slot di Insegnamenti appartenenti allo stesso Orientamento non possono

[18] La stessa equazione è implementata anche invertendo s1 e s2.

[19] Viene considerata la non intersezione tra i Docenti titolari dei differenti Slot poichè in tal caso si cadrebbe nel vincolo descritto dalla (5.41).

sovrapporsi come soft o hard constraint seguendo la Tabella 3.1:

$$\begin{gathered} o \in O, \forall i1, i2 \in \mathit{insOrientamento(o)}, i1 \neq i2, \\ \forall s1 \in \mathit{slotInsegnamento(i1)}, \forall s2 \in \mathit{slotInsegnamento(i2)}, \\ \mathit{se} : \mathit{periodoDidattico(i1,o)} = \mathit{periodoDidattico(i2,o)} \wedge \\ (\mathit{alfabetica(i1,o)} = \mathit{alfabetica(i2,o)} \vee \\ \mathit{alfabetica(i1,o)} = NoAlfabetica \vee \mathit{alfabetica(i2,o)} = NoAlfabetica) \wedge \\ (\mathit{squadraSlot(s1)} = NoSquadra \vee \\ \mathit{squadraSlot(s2)} = NoSquadra \vee \mathit{squadraSlot(s1)} = \mathit{squadraSlot(s2)}) \end{gathered} \tag{5.46}$$

$$\begin{gathered} (5.46) \ \wedge \mathit{penalitaSovr(i1,i2,o)} = LV\_H \wedge X\_d_{s1} = X\_d_{s2} \Rightarrow \\ (X\_h_{s1} \geq X\_h_{s2} + \mathit{numSlot(s2)} \vee X\_h_{s2} \geq X\_h_{s1} + \mathit{numSlot(s1)}) \end{gathered} \tag{5.47}$$

$$\begin{gathered} (5.46) \ \wedge \mathit{penalitaSovr(i1,i2,o)} \neq LV\_H \wedge X\_d_{s1} = X\_d_{s2} \wedge \\ \neg(X\_h_{s1} \geq X\_h_{s2} + \mathit{numSlot(s2)}) \wedge \\ \neg(X\_h_{s2} \geq X\_h_{s1} + \mathit{numSlot(s1)}) \Rightarrow \\ X\_\mathit{penalties}_{s1,s2,o} = \mathit{penalitaSovr(i1,i2,o)} \end{gathered} \tag{5.48}$$

Vengono aggiunte al modello le Equazioni (5.47) e (5.48).

- Preferenza su quando allocare o non allocare gli Slot di un Insegnamento i passato come parametro seguendo l'esempio 11[20]:

$$\begin{gathered} \forall s \in \mathit{slotInsegnamento(i)}, \\ \mathit{se} : X\_d_s = Lun \Rightarrow \\ (X\_h_s > 8.30 - 10.00 \vee X\_h_s + \mathit{numSlot(s)} \leq 8.30 - 10.00) \end{gathered} \tag{5.49}$$

[20]Data la possibilità di specificare diverse combinazioni possibili a livello di preferenza (favorevole, contrario) e di logica temporale (primaDi, ugualeA, dopoDi) vengono riportati a titolo esemplificativo i casi di riga 9 e 8 dell'Esempio 11. In questo modo vengono modellate le indisponibilità dei Docenti.

$$\begin{gathered}\forall s \in \textit{slotInsegnamento(i)}, \\ se : X\_d_s = Lun \wedge \\ \neg(X\_h_s > 17.30 - 19.00 \vee X\_h_s + \textit{numSlot(s)} \leq 17.30 - 19.00) \\ \Rightarrow X\_penalties_{prefI} = LV\_2\end{gathered} \tag{5.50}$$

- Associazione degli slot orari di lezione degli Insegnamenti appartenenti ad un dato Orientamento ed un dato Anno Accademico a X_ORIENT:

$$\begin{gathered}o \in O, a \in [1,3], i \in \textit{kernelInsInAnnoOrientamento(o,a)}, \\ \forall s \in \textit{slotInsegnamento(i)}, \forall g \in G, \forall f \in F, \forall n \in [0, \textit{numSlot(s)},] \\ se : X\_d_s = g \wedge X\_h_s = f \\ \Rightarrow X\_orient_{o,a,g,f+n} = 1\end{gathered} \tag{5.51}$$

- Per ogni Anno Accademico di ogni Orientamento, non devono esserci più di 6 slot orari giornalieri allocati per Insegnamenti di tipo Obbligatorio, Obbligatorio a scelta e Suggerito:

$$o \in O, a \in [1,3], g \in G, \sum_{f \in F} X\_orient_{o,a,g,f} \leq 6 \tag{5.52}$$

- Per ogni Anno Accademico di ogni Orientamento, non devono esserci più di 5 slot orari consecutivi allocati lo stesso giorno per Insegnamenti di tipo Obbligatorio, Obbligatorio a scelta e Suggerito:

$$\begin{gathered}o \in O, a \in [1,3], g \in G, \\ se : \sum_{f \in F} X\_orient_{o,a,g,f} = 6 \Rightarrow \\ X\_orient_{o,a,g,0} = 1 \wedge X\_orient_{o,a,g,6} = 1\end{gathered} \tag{5.53}$$

- Per ogni Anno Accademico di ogni Orientamento, se ci sono 4 slot orari consecutivi allocati lo stesso giorno per Insegnamenti di tipo Obbligatorio, Obbligatorio a scelta e Suggerito allora viene assegnata una penalità

pari a LV_1[21]:

$$\begin{gathered} o \in O, a \in [1,3], g \in G, f1 \in [0,3], \\ se: \sum_{0+f1 \leq f < f1+4} X\_orient_{o,a,g,f} = 4 \Rightarrow \\ X\_penaltiesStud_{pStud} = LV\_1 \end{gathered} \tag{5.54}$$

- Per ogni Anno Accademico di ogni Orientamento, se c'è un intervallo di 2 slot orari consecutivi lo stesso giorno per Insegnamenti di tipo Obbligatorio, Obbligatorio a scelta e Suggerito allora viene assegnata una penalità pari a LV_1[22]:

$$\begin{gathered} o \in O, a \in [1,3], g \in G, f1 \in [1,4], \\ se: \sum_{0 \leq f < f1} X\_orient_{o,a,g,f} = \sum_{0 \leq f < f1+2} X\_orient_{o,a,g,f} \wedge \\ \sum_{0 \leq f < f1} X\_orient_{o,a,g,f} \neq \sum_{0 \leq f < f1+3} X\_orient_{o,a,g,f} \Rightarrow \\ X\_penaltiesStud_{pStud} = LV\_1 \end{gathered} \tag{5.55}$$

- Per ogni Anno Accademico di ogni Orientamento, se ci sono 2 intervalli lo stesso giorno per Insegnamenti di tipo Obbligatorio, Obbligatorio a scelta e Suggerito allora viene assegnata una penalità pari a LV_6:

$$\begin{gathered} o \in O, a \in [1,3], g \in G, f1 \in [1,3], f2 \in [f1+2, 5], \\ se: \sum_{0 \leq f < f1} X\_orient_{o,a,g,f} = \sum_{0 \leq f < f1+1} X\_orient_{o,a,g,f} \wedge \\ \sum_{0 \leq f < f1} X\_orient_{o,a,g,f} \neq \sum_{0 \leq f < f2} X\_orient_{o,a,g,f} \wedge \\ \sum_{0 \leq f < f2} X\_orient_{o,a,g,f} = \sum_{0 \leq f < f2+1} X\_orient_{o,a,g,f} \wedge \\ \sum_{0 \leq f < f2} X\_orient_{o,a,g,f} \neq \sum_{f \in F} X\_orient_{o,a,g,f} \wedge \\ \Rightarrow X\_penaltiesStud_{pStud} = LV\_6 \end{gathered} \tag{5.56}$$

[21]Lo stesso vincolo è modellato su un limite di 5 slot orari consecutivi con penalita pari a LV_2.

[22]Lo stesso vincolo è modellato su un intervallo di più di 2 slot orari con penalità pari a LV_4

- In ogni slot orario della settimana gli Slot allocati per un determinato tipo di Locale e per la relativa capienza devono essere minori o uguali di quelle disponibili:

$$W(s,g,f,c) = \begin{cases} & \textit{tipoLocale(s)} = l \wedge \textit{capienzaLocale(s)} = c \wedge \\ 1 \quad \text{se} & X\_d_s = g \wedge X\_h_s \leq f \wedge \\ & X\_h_s + \textit{numSlot(s)} > f \\ 0 \quad \text{altrimenti} \end{cases}$$

$$\begin{gathered} g \in G, f \in F, c \in C, l \in L, \\ \sum_{s \in S} W(s,g,f,c) \leq \textit{numLocaliDisponibili(g,f,l,c)} \end{gathered} \tag{5.57}$$

- In un giorno il numero di slot orari associati ad un Docente è limitato superiormente:

$$Y(g,s) = \begin{cases} \textit{numSlot(s)} & \text{se X\_d\_s = g} \\ 0 & \text{altrimenti} \end{cases}$$

$$\begin{gathered} d \in D, g \in G, \\ \sum_{s \in \textit{slotDocente(d)}} Y(g,s) \leq \textit{numSlotDocente(d)} \end{gathered} \tag{5.58}$$

- Preferenza su quando allocare o non allocare gli Slot di un Docente d passato come parametro seguendo l'esempio 12: stessa struttura delle equazioni (5.49) e (5.50)

- Distanza minima tra due Slot associati ad uno stesso Docente d ad una distanza minima $dist \in \mathbb{N}$[23]. Può essere trattato come hard constraint (5.59) o come soft constraint (5.60):

$$\begin{gathered} s1, s2 \in \textit{slotDocente(d)}, \textit{se} X\_d_{s1} = X\_d_{s2} \Rightarrow \\ X\_h_{s1} \geq X\_h_{s2} + \textit{numSlot(s2)} + \textit{dist} \vee \\ X\_h_{s2} \geq X\_h_{s1} + \textit{numSlot(s1)} + \textit{dist} \end{gathered} \tag{5.59}$$

[23]Questo vincolo non è attivato di default, per attivarlo va definito come nell'Esempio 16.

$$\begin{gathered} s1, s2 \in slotDocente(d), seX\_d_{s1} = X\_d_{s2} \wedge \\ \neg(X\_h_{s1} \geq X\_h_{s2} + numSlot(s2) + dist \vee \\ X\_h_{s2} \geq X\_h_{s1} + numSlot(s1) + dist) \Rightarrow \\ X\_penaltiesDoc_{pDoc} = LV\_* \end{gathered} \tag{5.60}$$

- Distanza massima tra due Slot associati ad uno stesso Docente d ad una distanza massima $dist \in \mathbb{N}$[24]. Può essere trattato come hard constraint (5.61) o come soft constraint (5.62):

$$\begin{gathered} s1, s2 \in slotDocente(d), \\ \\ se : X\_d_{s1} = X\_d_{s2} \wedge X\_h_{s1} > X\_h_{s2} \\ \Rightarrow X\_h_{s1} \leq X\_h_{s2} + numSlot(s2) + dist \\ se : X\_d_{s1} = X\_d_{s2} \wedge X\_h_{s2} > X\_h_{s1} \\ \Rightarrow X\_h_{s2} \leq X\_h_{s1} + numSlot(s1) + dist \end{gathered} \tag{5.61}$$

$$\begin{gathered} s1, s2 \in slotDocente(d), \\ \\ se : X\_d_{s1} = X\_d_{s2} \wedge (\\ (X\_h_{s1} > X\_h_{s2} \wedge X\_h_{s1} > X\_h_{s2} + numSlot(s2) + dist) \vee \\ (X\_h_{s2} > X\_h_{s1} \wedge X\_h_{s2} > X\_h_{s1} + numSlot(s1) + dist)) \\ \Rightarrow X\_penaltiesDoc_{pDoc} = LV\_* \\ se : \neg(X\_d_{s1} = X\_d_{s2} \wedge (\\ (X\_h_{s1} > X\_h_{s2} \wedge X\_h_{s1} > X\_h_{s2} + numSlot(s2) + dist) \vee \\ (X\_h_{s2} > X\_h_{s1} \wedge X\_h_{s2} > X\_h_{s1} + numSlot(s1) + dist))) \\ \Rightarrow X\_penaltiesDoc_{pDoc} = 0 \end{gathered} \tag{5.62}$$

- Slot associati ad uno stesso Docente d e appartenenti ad Insegnamenti diversi non dovrebbero essere allocati lo stesso giorno[25]. Può essere

[24] Questo vincolo non è attivato di default, per attivarlo va definito come nell'esempio 16.

[25] Questo vincolo non è attivato di default, per attivarlo va definito come nell'esempio 16.

trattato come hard constraint (5.63) o come soft constraint (5.64):

$$s1, s2 \in slotDocente(d),$$
$$X\_d_{s1} \neq X\_d_{s2} \tag{5.63}$$

$$s1, s2 \in slotDocente(d),$$
$$se : X\_d_{s1} = X\_d_{s2} \Rightarrow X\_penaltiesDoc_{pDoc} = LV\_* \tag{5.64}$$
$$se : X\_d_{s1} \neq X\_d_{s2} \Rightarrow X\_penaltiesDoc_{pDoc} = 0$$

- Slot s1 almeno un giono dopo un altro Slot s2. Può essere trattato come hard constraint (5.65) o come soft constraint (5.66):

$$X\_d_{s1} > X\_d_{s2} \tag{5.65}$$

$$se : X\_d_{s1} > X\_d_{s2} \Rightarrow X\_penaltiesDoc_{pDoc} = -LV\_* \tag{5.66}$$
$$se : \neg(X\_d_{s1} > X\_d_{s2}) \Rightarrow X\_penaltiesDoc_{pDoc} = 0$$

- Slot s1 in un giorno diverso da un altro Slot s2. Può essere trattato come hard constraint (5.67) o come soft constraint (5.68):

$$X\_d_{s1} \neq X\_d_{s2} \tag{5.67}$$

$$se : X\_d_{s1} \neq X\_d_{s2} \Rightarrow X\_penaltiesDoc_{pDoc} = -LV\_* \tag{5.68}$$
$$se : \neg(X\_d_{s1} \neq X\_d_{s2}) \Rightarrow X\_penaltiesDoc_{pDoc} = 0$$

- Slot s1 stesso giorno di un altro Slot s2. Può essere trattato come hard constraint (5.69) o come soft constraint (5.70):

$$X\_d_{s1} = X\_d_{s2} \tag{5.69}$$

$$se : X\_d_{s1} = X\_d_{s2} \Rightarrow X\_penaltiesDoc_{pDoc} = -LV\_* \tag{5.70}$$
$$se : \neg(X\_d_{s1} = X\_d_{s2}) \Rightarrow X\_penaltiesDoc_{pDoc} = 0$$

- Slot s1 parallelo ad un altro Slot s2. Può essere trattato come hard constraint (5.71) o come soft constraint (5.72):

$$X\_d_{s1} = X\_d_{s2} \wedge X\_h_{s1} = \wedge X\_f_{s2} \tag{5.71}$$

$$\begin{gathered} se : X\_d_{s1} = X\_d_{s2} \wedge X\_h_{s1} = \wedge X\_f_{s2} \\ \Rightarrow X\_penaltiesDoc_{pDoc} = -LV\_* \\ se : \neg(X\_d_{s1} = X\_d_{s2} \wedge X\_h_{s1} = \wedge X\_f_{s2}) \\ \Rightarrow X\_penaltiesDoc_{pDoc} = 0 \end{gathered} \tag{5.72}$$

- Slot s1 subito dopo un altro Slot s2. Può essere trattato come hard constraint (5.73) o come soft constraint (5.74):

$$X\_d_{s1} = X\_d_{s2} \wedge X\_h_{s1} = X\_h_{s2} + numSlot(s2) \tag{5.73}$$

$$\begin{gathered} se : X\_d_{s1} = X\_d_{s2} \wedge X\_h_{s1} = X\_h_{s2} + numSlot(s2) \\ \Rightarrow X\_penaltiesDoc_{pDoc} = -LV\_* \\ se : \neg(X\_d_{s1} = X\_d_{s2} \wedge X\_h_{s1} = X\_h_{s2} + numSlot(s2)) \\ \Rightarrow X\_penaltiesDoc_{pDoc} = 0 \end{gathered} \tag{5.74}$$

5.3.2 Modellazione con equazioni

La rappresentazione in Python di tutte le equazioni presenti nel modello sono riportate in Appendice A.

5.3.3 Conclusioni

Questo modello consente di modellare tutti i requisiti di alto livello ad eccezione della gestione della distanza tra Locali. Inoltre le performance ottenute con il software CPLEX garantiscono un elevato grado di scalabilità con un aumento accettabile delle risorse necessarie.

Capitolo 6

Modi di utilizzo del tool

Ci sono quattro modalità di utilizzo sel software: validazione, ottimizzazione globale, ottimizzazione bilanciata, ottimizzazione incrementale.

6.1 Validazione

In questa modalità vengono considerati esclusivamente gli Hard Constraint senza tenere conto della Funzione obiettivo (5.19). Operando in questo modo viene istanziata una sottoparte del modello escludendo $X\_penalties_p$, $X\_penaltiesStud_{pStud}$ e $X\_penaltiesDoc_{pDoc}$ al fine di ottimizzare le performance. L'obiettivo di questa modalità è di ottenere in un tempo minore una risposta positiva/negativa circa la presenza di almeno una soluzione per i dati correnti[1].

In fase di lancio del tool vengono configurati tutte le sorgenti di dato:

- *add_folderInsegnamentiXml()* per indicare la cartella contenente i file XML con i Template Orario come quello dell'Esempio 4.2.2
- *set_jsonFileDocenti()* per configurare il file JSON con le preferenze dei Docenti come negli Esempi 12, 13, 14, 15 e 16
- *set_jsonFileInsegnamenti()* per configurare il file JSON con le preferenze degli Insegnamenti come nell'Esempio 11

[1]Di norma andrebbe prima lanciata una simulazione in questa modalità e soltanto successivamente in una delle modalità con ottimizzazione.

- *set_fileXmlLocaliAllocati()* per caricare il file XML con i dati dei Locali disponibili come nell'Esempio 10.

- *set_nomeSolutions()* per configurare il nome con cui viene salvata la soluzione nel db[2]

- *set_capienzaAuleExtended()* per indicare di quanti livelli di capienza dell'Aula superiore a quella necessaria sia possibile allocare un'aula.

Di seguito la funzione di lancio della simulazione in modalità validazione:

```
def checkValiditySolution():
    '''Non viene attivato nessun soft constraint, l'obiettivo è solamente
     verificare la validità della sol proposta. E' utile in caso
    si voglia mantenere buona parte del modello da una precedente
    allocazione e predisporre una particolare configurazione per certi
    Insegnamenti o se si vuole testare la compatibilità di hard
    constraint diversi'''
    p1 = Parameter()
    p1.add_folderInsegnamentiXml(basePath + "/data/nuovi_XML")
    p1.set_jsonFileDocenti(basePath + "/data/nuovo_docenti.json")
    p1.set_jsonFileInsegnamenti(basePath + "/data/nuovo_insegnamenti.json
    ")
    p1.set_fileXmlLocaliAllocati(basePath + "/data/nuove_aule.xml")
    p1.set_nomeSolutions("sim_validation")
    logs.startCall()

    p1.set_capienzaAuleExtended(0)
```

6.2 Ottimizzazione globale

In questa modalità viene attivata l'Equazione (5.19) considerando tutti i Soft Constraint e tutti gli Hard Constraint; la funzione obiettivo risulta essere componibile in modo da poter anche considerare un sottoinsieme delle

[2]Il nome indicato sarà il prefisso della soluzione seguito da "_" e un numero corrispondente al valore della funzione obiettivo per tale soluzione o -1 in caso di lancio in modalità validazione.

penalità. Converge alla soluzione matematicamente ottima ma richiede tempi di esecuzione elevati rispetto alla modalità validazione[3].

Viene aggiunto un comando per indicare i componenti della funzione obiettivo da ottimizzare tramite chiamate multiple alla funzione *add_optimizeComponent()*, i valori del parametro rappresentano:

- Parameter.OptimizeComponent.Orientamento: minimizzazione delle penalità relative agli Orientamenti come la sovrapposizione degli Insegnamenti quali quelli di tipo Credito libero, Insegnamento a scelta
- Parameter.OptimizeComponent.Studenti: minimizzazione delle penalità relative alla bontà dell'orario in termini di spazi vuoti e slot consecutivi per gli Insegnamenti di un dato Orientamento
- Parameter.OptimizeComponent.Docenti: minimizzazione delle penalità rispetto a tutte le preferenze espresse dai Docenti.

Di seguito la funzione di lancio della simulazione in modalità ottimizzazione globale:

```
1  def computeOptimalSolution():
2      '''Simile a computeOptimalSolution(). Solo non impongo vincoli
       ausiliari sulle varie parti di soft constraint (ie non mi interessa
3      se la penalità per un Docente sarà superiora alla media, e così via),
        mi basta puntare a minimizzare la funzione obiettivo.\n
4      Potrebbe portare risultati praticamente analoghi a
       computeOptimalSolution() con meno effort'''
5      p1 = Parameter()
6      p1.add_folderInsegnamentiXml(basePath + "/data/nuovi_XML")
7      p1.set_jsonFileDocenti(basePath + "/data/nuovo_docenti.json")
8      p1.set_jsonFileInsegnamenti(basePath + "/data/nuovo_insegnamenti.json
       ")
9      p1.set_fileXmlLocaliAllocati(basePath + "/data/nuove_aule.xml")
10     p1.set_nomeSolutions("sim_globalOptimization")
11     logs.startCall()
12
13     p1.set_capienzaAuleExtended(0)
```

[3]Andrebbe usata solo successivamente ad una simulazione in modalità validazione con esito positivo.

```
14
15      p1.add_optimizeComponent(Parameter.OptimizeComponent.Orientamento)
16      p1.add_optimizeComponent(Parameter.OptimizeComponent.Studenti)
17      p1.add_optimizeComponent(Parameter.OptimizeComponent.Docenti)
```

6.3 Ottimizzazione bilanciata

Coincide con quanto detto per l'ottimizzazione globale, andando inoltre a bilanciare le penalità associate ai vari Docenti al fine di non ottenere soluzioni ottime dal punto di vista di certi Docenti ma pessime per altri: non è affatto scontato che la soluzione matematicamente ottima sia anche quella migliore adottabile.

Il comando *p1.add_additionalModelConfig()* con parametro Parameter.OptimizeComponent.Docenti serve per bilanciare le penalità relative ai Docenti, in maniera analoga è possibile bilanciare le penalità relative agli Orientamenti e agli studenti utilizzando rispettivamente i parametri Parameter.OptimizeComponent.Orientamento e Parameter.OptimizeComponent.Studenti[4].

Di seguito la funzione di lancio della simulazione in modalità ottimizzazione bilanciata:

```
1  def computeOptimalSolutionBalanced():
2      '''Abilita tutti i meccanismi di penalità al fine di trovare la
       soluzione ottima'''
3      p1 = Parameter()
4      p1.add_folderInsegnamentiXml(basePath + "/data/nuovi_XML")
5      p1.set_jsonFileDocenti(basePath + "/data/nuovo_docenti.json")
6      p1.set_jsonFileInsegnamenti(basePath + "/data/nuovo_insegnamenti.json
       ")
7      p1.set_fileXmlLocaliAllocati(basePath + "/data/nuove_aule.xml")
8      p1.set_nomeSolutions("sim_balancedOptimization")
9      logs.startCall()
10
11     p1.add_optimizeComponent(Parameter.OptimizeComponent.Orientamento)
12     p1.add_optimizeComponent(Parameter.OptimizeComponent.Studenti)
```

[4]Non è necessario utilizzarli in mutua esclusione.

```
13      p1.add_optimizeComponent(Parameter.OptimizeComponent.Docenti)
14
15      # bilanciamento dei Docenti
16      p1.add_additionalModelConfig(Parameter.OptimizeComponent.Docenti)
```

6.4 Ottimizzazione incrementale

Questa modalità funziona esattamente come l'Ottimizzazione globale utilizzando come base una precedente soluzione ottenuta con una delle modalità precedenti. Vengono quindi definiti esplicitamente gli Insegnamenti[5] per cui verranno ignorati i dati nella soluzione precedente e che andranno nuovamente allocati.

- Il comando *set_usePianoAllocazioneAsBase(True)* abilita la modalità
- Il comando *p1.set_pianoAllocazioneBase()* configura la soluzione a db da usare come sorgente
- Il comando *add_listID_INCToModify()* indica gli unici Insegnamenti per cui procedere alla riallocazione dell'orario.

Di seguito la funzione di lancio della simulazione in modalità ottimizzazione incrementale:

```
1  def keepOldModelWithMinorChanges_optimize():
2      '''Uso un piano allocazione precedentemente creato per buona parte
       degli Insegnamenti, solo alcuni di essi devono essere modificati,
3      voglio trovare la soluzione ottima.'''
4      p1 = Parameter()
5      p1.add_folderInsegnamentiXml(basePath + "/data/nuovi_XML")
6      p1.set_jsonFileDocenti(basePath + "/data/nuovo_docenti.json")
7      p1.set_jsonFileInsegnamenti(basePath + "/data/nuovo_insegnamenti.json
       ")
8      p1.set_fileXmlLocaliAllocati(basePath + "/data/nuove_aule.xml")
9      p1.set_nomeSolutions("sim_incrementalOptimization")
10     logs.startCall()
```

[5] Tramite l'ID_INC.

```
11
12      p1.set_capienzaAuleExtended(0)
13
14      p1.add_optimizeComponent(Parameter.OptimizeComponent.Orientamento)
15      p1.add_optimizeComponent(Parameter.OptimizeComponent.Studenti)
16      p1.add_optimizeComponent(Parameter.OptimizeComponent.Docenti)
17
18      p1.set_usePianoAllocazioneAsBase(True)
19      p1.set_pianoAllocazioneBase("sim_globalOptimization")
20      p1.add_listID_INCToModify([259328,261125])
```

Capitolo 7

Testing e validazione

7.1 Validazione preliminare con dati anno precedente

Per poter validare il funzionamento del tool è stato necessario utilizzare un primo dataset di testing in attesa di recepire i dati ufficiali ottenuti solo successivamente. Per tale scopo sono stati usati i dati del primo semestre dell'Anno Accademico 2021-2022 poiché molto simili a quelli su cui operare successivamente[1].

7.1.1 Dati provenienti dall'Ufficio Gestione Aule Anno Accademico 2021-2022

Sorgente principale di dati per questa prima validazione è il documento pervenuto dall'Ufficio Gestione Aule, Area Edilizia e Logistica (EDILOG) del Politecnico di Torino, contenente per ogni Orientamente dei vari Cdl in capo ai collegi ICM e ETF gli Insegnamenti che ne fanno parte di cui è presente un estratto in Figura 7.1. Da questo file sono stati estratti per ogni Orientamento tutti gli Insegnamenti con il rispettivo TipoInsegnamento ed inseriti a db secondo il formato definito nella sezione 4.2.4 tramite script Python.

[1]Mediamente variano 10/20 corsi tra un Anno Accademico e l'altro, con piccole variazioni all'interno della pianificazione dei vari Orientamenti.

	G	H	I	J	K
1	COD_INS	TITOLO	ID_INC	ANNO	ORIENTAMENTO/GRUPPO
81	15CDUPC	Reti di calcolatori	263360	3	Introductory year taught in Englis
82	15CDUPC	Reti di calcolatori	263360	3	Percorso (Insegnamento a scelta da Tabella F)
83	15CDUPC	Reti di calcolatori	263360	3	Percorso INTRAPRENDENTI (Insegnamento a scelta da Tabella F)
84	15CDUPC	Reti di calcolatori	263360	3	Primo anno sede di Mondovì (Insegnamento a scelta da Tabella F)

Figura 7.1: Estratto documento dell'Ufficio Gestione Aule.

Il file è stato anche la sorgente per la definizione dei file XML degli Insegnamenti secondo lo schema Template Orario introdotto nella Sezione 4.2.1 dato che si trattava di dati inediti e non previsti fino allo scorso anno; anch'essi sono stati generati tramite script Python.

7.1.2 Dati estratti dal sito del Politecnico

Da una prima analisi dei dati estratti come descritto nella Sezione 7.1.1 è chiaro come risulti ambiguo ricavare il TipoInsegnamento di un dato Insegnamento all'interno di un Orientamento.

Le informazioni mancanti sono estratte direttamente dai piani studi dei vari Orientamenti presenti sul sito del Politecnico.

L'importazione dei dati da tale fonte è avvenuta mediante la scrittura di un web scraper scritto in Python avvalendosi delle librerie Requests[2] e BeatifulSoup[3]. Di seguito vengono mostrati i risultati ottenuti in seguito ad alcune operazioni manuali volte ad uniformare i dati ricavati :

```
['cdl', 'Corso  di Laurea in Ingegneria  fisica (Torino)', 'https://
    didattica.polito.it/pls/portal30/sviluppo.offerta_formativa_2019.vis?
    p_coorte=2022&p_sdu=37&p_cds=9']
,['orientamento', 'Percorso']
,['table_title', '1 anno 2021/2022']
```

[2]https://pypi.org/project/requests/

[3]https://beautiful-soup-4.readthedocs.io/en/latest/

```
4 ,['table_content', [['periodo', 'codIns', 'cfu'], ['1', '16ACFOD', '
    Analisi matematica I'], ['1', '15AHMOD', 'Chimica'], ['1', '14BHDOD',
    'Informatica'], ['1; 2', '07LKIOD', 'Lingua inglese I livello'], ['2',
     '01RKCOD', 'Algebra lineare e geometria'], ['2', '17AXOOD', 'Fisica I
    '], ['2', '04BKTOD', 'Laboratorio di fisica']]]
5 ,['table_title', '2 anno 2022/2023']
6 ,['table_content', [['periodo', 'codIns', 'cfu'], ['1,2', '', ''], ['
    oppure'], ['1,2', '', ''], ['2', '01USBOD', '']]]
7 ,['table_title', '3 anno 2023/2024']
8 ,['table_content', [['periodo', 'codIns', 'cfu'], ['1', '02NKUOD', '
    Elettromagnetismo applicato'], ['1', '06AXFOD', 'Fisica dello stato
    solido'], ['1', '02MZGOD', 'Applied electronics'], ['oppure'], ['1', '
    07ATIOD', 'Elettronica applicata'], ['1,2', 'Insegnamento a scelta da
    Tabella B', '6'], ['1,2', '32IBNOD', ''], ['2', 'Insegnamento a scelta
     da Tabella 1', '8'], ['1,2', 'Crediti liberi del 3 anno', '6'], ['2',
     '01NTQOD', 'Tecnologie per le nanoscienze']]]
```

Un'ulteriore informazione importata dal sito del Politecnico è il numero di studenti iscritti ad un determinato Insegnamento, avvenuto anch'esso tramite un altro web scraper scritto in Python mediante l'ausilio della libreria Requests in grado di ricavare i dati desiderati sulla base dei superi degli esami.

7.1.3 Problemi di performance

Sulla base del dataset realizzato vengono fatti dei test al fine di valutare le performance del tool inizialmente in ambiente locale[4] osservando come in modalità validazione restave in esecuzione per diversi giorni senza restituire soluzioni mentre in modalità ottimizzazione restituiva un eccezione in seguito ad una completa saturazione della memoria disponibile.

In seguito a richiesta all'Area IT del Politecnico di Torino viene messo a disposizione un nodo di calcolo interno al Politecnico disponente di 64 core e 192 GB di RAM. Con questa nuova configurazione il tempo di esecuzione del software in modalità validazione è di qualche giorno mentre è indefinito in ottimizzazione [5]. Da prove sperimentali successive emerge che tali tempistiche

[4]L'ambiente locale prevede un PC portatile con processore Intel Core i7 (4 core) e 12 GB di RAM.

[5]Il tool è stato lasciato in esecuzione per circa tre settimane senza che iniziasse la fase

siano dovute a dei limiti particolarmente stretti circa le aule disponibili.

7.1.4 Visualizzazione grafica dei risultati

La validazione preliminare dei risultati è uno scoglio non indifferente a causa della mole di essi e dei vincoli a cui i vari Slot devono sottostare. Viene quindi implementata una web app minimale mediante la libreria React[6] che permette di filtrare gli Slot di una soluzione generata per CDL, Orientamento e Periodo didattico come mostrato in Figura 7.2.

Questa applicazione permette una visualizzazione tabellare dell'orario set-

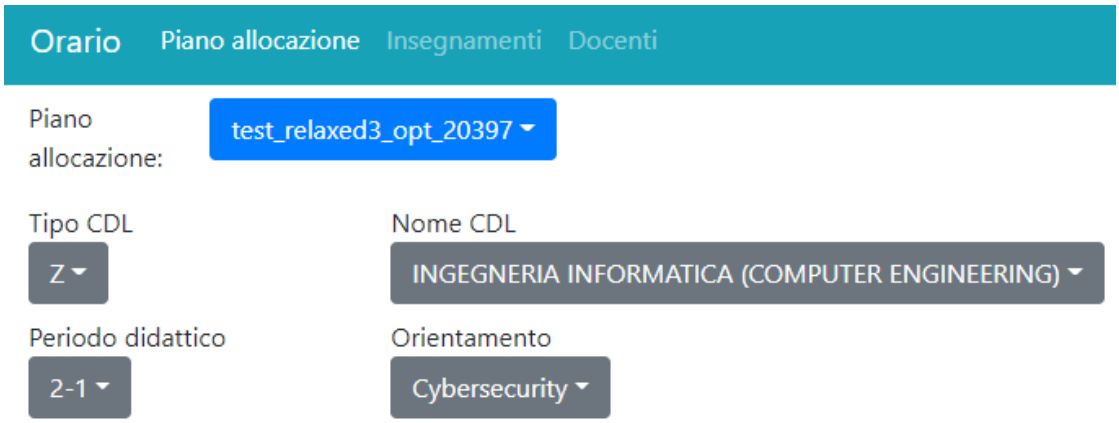


Figura 7.2: Finestra di configurazione dell'applicazione per la visualizzazione grafica dei risultati.

timanale di un dato Orientamento, mostrando in ogni slot orario i vari Slot allocati; inoltre prevede colorazioni differenti a seconda che si tratti di un Insegnamento obbligatorio, obbligatorio a scelta o suggerito (rosso), un Insegnamento a scelta (giallo) o un credito libero (verde) come mostrato in Figura 7.3 permettendo di individuare immediatamente eventuali sovrapposizioni.

di convergenza verso la soluzione ottima

[6]https://it.reactjs.org/

test_relaxed3_opt_20397	Lunedì	Martedì
8.30-10.00		
10.00-11.30		Computational inte 260461_slot3
11.30-13.00	Progettazione di s 259940_slot1	
13.00-14.30	Modern design of c 259421_slot_lab1 Progettazione di s 259940_slot1 Security verificat 260188_slot1 Computational inte 260461_slot1	GPU programming 260466_slot1 Human Computer Int 261102_slot2

Figura 7.3: Estratto orario applicazioneper la visualizzazione grafica dei risultati.

7.2 Acquisizione dati Anno Accademico 2022-2023

L'acquisizione dei nuovi dati ha comportato l'affrontare alcune nuove problematiche che verranno discusse qui di seguito.

7.2.1 Dati provenienti dall'Ufficio Gestione Aule

Come per i dati di prova vengono usati i dati dell'Ufficio Gestione Aule per ricavare gli Insegnamenti associati ai vari Orientamenti, tuttavia non vengono più generati i file XML degli Insegnamenti a partire da questa fonte. Questi dati vengono arricchiti con i dati presenti sul sito del Politecnico come descritto nella Sezione 7.1.2.

L'ufficio Gestione Aule stabilisce, compatibilmente con gli Insegnamenti erogati dagli altri collegi, il numero, il tipo e la capienza delle aule e dei laboratori resi disponibili per l'allocazione dei due collegi in esame. Questo

documento è stato trasformato manualmente in XML seguendo lo schema introdotto in fase di progettazione nella Sezione 4.2.2

7.2.2 Dati supplementari inseriti manualmente

Dai risultati ottenuti dai primi tentativi in modalità validazione sono emersi alcuni problemi di dati mancanti con le sorgenti fin qui considerate, in particolare riguardo gli Orientamenti in cui compaiono gli Insegnamenti, nonchè gli Insegnamenti veri e propri di cui bisogna allocare l'Orario. La soluzione adottata prevede il completamento del database con i dati mancanti inseriti manualmente come mostrato qui di seguito[7]:

```
# Inserimenti Orientamenti puntuali

ins:List[Tuple[str,TipoInsegnamento]] = list()
ins.append(("261457", TipoInsegnamento.Obbligatorio))
ins.append(("261456", TipoInsegnamento.Obbligatorio))
ins.append(("258956", TipoInsegnamento.Obbligatorio))
ins.append(("258955", TipoInsegnamento.Obbligatorio))
ins.append(("258954", TipoInsegnamento.Obbligatorio))
db.update_InsegnamentiInOrientamento("Percorso", "COMMUNICATIONS
    ENGINEERING", ins, 'Z', '1-1')

ins:List[Tuple[str,TipoInsegnamento]] = list()
ins.append(("260275", TipoInsegnamento.Obbligatorio))
db.add_InsegnamentiInOrientamento("MNIS - Micro and Nanotechnologies for
    Integrated Systems", "NANOTECHNOLOGIES FOR ICTs (NANOTECNOLOGIE PER LE
     ICT)", 'Z', '1-1', ins)
```

Un altro affinamento dei dati è stato fatto per consentire un'allocazione anche parallela delle istanze in lingua italiana ed inglese di uno stesso Insegnamento, seppure venissero considerare entrambi di tipo Obbligatorio, Obbligatorio a scelta o Suggerito per un dato Orientamento. Viene quindi definita la seguente struttura ausiliaria come sorgente di dati per il tool di cui è riportato un estratto di codice:

[7]Purtroppo qualche sporadico errore è rimasto anche successivamente data la mole di dati. Il codice mostrato è solamente un estratto dei dati aggiunti.

```
list_ID_INC_parallelizzabili = []
'''Ogni entry contiene gli ID_INC di corsi parallelizzabili tra loro anche se appartenenti allo stesso Orientamento
e Obbligatori (eg: alfabetiche diverse, Insegnamenti in italiano e in inglese).
Format: [[ID_INC_1, ID_INC_2], [.., ..], ..]'''
list_ID_INC_parallelizzabili.extend([['259206','260158', '259922'], ['259233','260158','259234'], ['259233','260302']])
```

Infine per completare la descrizione di Insegnamenti non sovrapponibili tra loro poichè non completa con i dati a disposizione, si è definita una nuova struttura in grado di modellare un conflitto binario tra Insegnamenti sia come soft che come hard constraint:

```
# permette di specificare coppie di Insegnamenti non sovrapponibili tra loro (a prescindere da tutti gli altri vincoli)
list_InsegnamentiNonSovrapponibiliCustom:List[Tuple[Tuple[int,int], LV_Penalties]] = list() # ((ID_INC1, ID_INC2), LV_PEN)

# Hard Constraints
list_InsegnamentiNonSovrapponibiliCustom.append(((261125,259419), LV_Penalties.LV_H))
list_InsegnamentiNonSovrapponibiliCustom.append(((259328,259419), LV_Penalties.LV_H))
# Soft Constraints
list_InsegnamentiNonSovrapponibiliCustom.append(((261103,259419), LV_Penalties.LV_5))
list_InsegnamentiNonSovrapponibiliCustom.append(((263355,259419), LV_Penalties.LV_5))
```

7.2.3 Jotform e desiderata Docenti

Per poter collezionare tutte le informazioni relative ai Docenti che descriverssero un'allocazione desiderata dell'orario e le loro preferenze in termini di indisponibilità è stato necessario introdurre un'ulteriore sorgente di dati che doveva risultare accessibile ai vari Docenti in modo immediato e consentire al tempo stesso un'estrazione in un tempo limitato dei file XML compatibili col formato descritto nella Sezione 4.2.1 e delle disponibilità dei

Docenti secondo il formato descritto nell'Esempio 11.

Per poter soddisfare tale compito si è fatto ricorso al tool disponibile online Jotform[8].

7.2.4 Insegnamenti appartenenti ai collegi ICM e ETF non presenti in Jotform

Per alcuni Orientamenti e alcuni Insegnamenti viene mantenuto l'orario dello scorso anno. Per far fronte a questa richiesta, vista l'impossibilità di utilizzare la modalità ottimizzazione incrementale poichè nel database non è presente l'allocazione dell'orario dello scorso anno, vengono generati manualmente i diversi file XML secondo lo schema Template Orario indicato nella Sezione 4.2.1.

7.2.5 Orari da altri collegi

Molti Docenti hanno lezioni appartenenti non solamente a Insegnamenti facenti capo ai collegi ICM e ETF di cui il tool deve elaborare l'orario, ma hanno lezioni anche in altri Insegnamenti di cui non se ne occupa il tool e che quindi devono essere descritti come slot orari preallocati per i Docenti.

Considerare tali Slot come indisponibilità dei Docenti non è corretto perchè sarebbe possibile allocare molti slot orari nella stessa giornata andando anche oltre il limite di slot orari giornalieri del Docente. Nemmeno considerarli SlotScelti del tipo definito fin qui non sarebbbe corretto poichè il tool allocherebbe anche i Locali necessari allo svolgimento di tali Slot.

La soluzione adottata prevede di creare Insegnamenti speciali caratterizzati da un ID_INC < 0 così da permettere al tool di gestirli correttamente:

```
1 <CatalogoTemplate xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="../../models/schemaUseCase.xsd">
2 <TemplateOrario><!-- generated by Francesco -->
3 <id>Insegnamenti Gestionali</id>
4 <ID_INC>-2</ID_INC>
5 <NumIscritti>0</NumIscritti>
```

[8]https://www.jotform.com/

```
6
7  <!-- imprenditorialità e innovazione -->
8  <SlotScelto slotId="slot3"><Docente>Landoni Paolo</Docente><
       NumSlotConsecutivi>1</NumSlotConsecutivi><Tipo>L</Tipo><Locale>Aula
       multimediale</Locale><Giorno>Lun</Giorno><FasciaOraria>14.30-16.00</
       FasciaOraria></SlotScelto>
9  <SlotScelto slotId="slot4"><Docente>Landoni Paolo</Docente><
       NumSlotConsecutivi>2</NumSlotConsecutivi><Tipo>L</Tipo><Locale>Aula
       multimediale</Locale><Giorno>Gio</Giorno><FasciaOraria>16.00-17.30</
       FasciaOraria></SlotScelto>
10
11 <!-- Basi di dati -->
12 <SlotScelto slotId="slot7"><Docente>Cagliero Luca</Docente><
       NumSlotConsecutivi>2</NumSlotConsecutivi><Tipo>L</Tipo><Locale>Aula</
       Locale><Giorno>Lun</Giorno><FasciaOraria>10.00-11.30</FasciaOraria></
       SlotScelto>
13 <SlotScelto slotId="slot8"><Docente>Cagliero Luca</Docente><
       NumSlotConsecutivi>2</NumSlotConsecutivi><Tipo>L</Tipo><Locale>Aula</
       Locale><Giorno>Gio</Giorno><FasciaOraria>13.00-14.30</FasciaOraria></
       SlotScelto>
14
15 </TemplateOrario>
16 </CatalogoTemplate>
```

7.3 Validazione dati Anno Accademico 2022-2023

7.3.1 Primi tentativi UNSAT

Le prime simulazioni lanciate in modalità validazione terminavano immediatamente poichè l'introduzione di molti Insegnamenti preallocati pregiudicava il soddisfacimento di tutti i vincoli definiti dai requisiti. In particolare è stato modificato parzialmente il tool per ignorare violazioni dei requisiti dovute ad un'approssimazione del modello reale nei seguenti casi:

- Slot di Insegnamenti di tipo Obbligatorio, Obbligatorio a scelta, Suggerito per un dato Insegamento paralleli tra loro in quanto la sovrapposizione risulta tollerata: viene sfruttata la struttura introdotta nella Sezione 7.2.2.
  Questo approccio non risulta corretto nel caso in cui i due Insegnamenti

siano composti anche da SlotGenerici, in quanto l'allentamento dei vincoli così definito ha valore a livello di Insegnamento e non di singolo Slot; viene quindi introdotta la possibilità di allentare il vincolo di non sovrapposizione con granularità maggiore agendo sul singolo Slot con l'implementazione di una nuova funzione:

```
def customCheckSlotParallelizzabili(self, slotId_i:str, slotId_j:str)
    -> bool:
    '''Controlla se due slot possono essere alloati in parallelo
    anche se secondo i normali requisiti non potrebbero'''

    # 259237 - Modeling and optimization of embedded systems Mar
    14:30-17:30 pd: PeriodoDidattico.PrimoAnno_PrimoSemestre - 259236
    - Electronics for embedded systems Mar 14:30-17:30 pd:
    PeriodoDidattico.PrimoAnno_PrimoSemestre
    if slotId_i in ["259237_slot_lab1", "259236_slot_lab2"] and
    slotId_j in ["259237_slot_lab1", "259236_slot_lab2"]:
        return True
```

- In certi casi venivano allocati quotidianamente più di 4 slot orari allo stesso Docente, si è quindi aumentato tale limite giornaliero per il Docente specifico come indicato nell'Esempio 13.

- Vi sono verificati casi di SlotScelti allocati in parallelo tenuti dallo stesso Docente: tale situazione è frutto di lezioni a settimane alterne all'interno di certi Insegnamenti che non creano veri conflitti. Si procede aggiungendo il suffisso "1" in uno dei due SlotScelti evitando di far considerare al tool allocati allo stesso Docente entrambi gli Slot.

- Esistono alcuni casi particolari di Insegnamenti in cui il semestre è suddiviso in più parti e tali parti sono erogate in periodi diversi del semestre a seconda dell'alfabetica poichè la singola porzione di lezioni è tenuta da un singolo Docente e questo non farà tutte le lezioni nello stesso periodo. La descrizione tramite XML per questi Insegnamenti non risulta sufficiente poichè imporrebbe che tutti questi Insegnamenti non siano allocati in parallelo in quanto tenuti da almeno un Docente in comune; l'approcio descritto al punto precedente anche non sarebbe corretto in quanto in caso gli Slot non fossero poi comunque allocati in parallelo ci troveremmo ad avere degli slot orari in cui tale Docente

risulta essere realmente indisponibile mentre così non sarebbe per il software.
L'approccio ideato prevede di inserire una nuova opzione come descritto nell'Esempio 14.

7.3.2 Allentamento vincoli con soft constraint

Tutti i raffinamenti dei dati messi in atto fino ad ora non sono stati sufficienti a garantire un'allocazione valida dell'orario a causa del numero elevato di vincoli sugli Insegnamenti appartenenti ai diversi Orientamenti definiti come Hard Constraint ed hai limiti fisici di disponibilità delle Aule. In seguito ad un allentamento di molti vincoli definiti precedentemente come Hard Constraint trasformandoli in Soft Constraint[9] la modalità validazione termina con esito positivo; lanciando il tool in modalità ottimizzazione è possibile ottenere allocazioni che garantiscano una sovrapposizione degli Insegnamenti relaviti ai vincoli allentati via via sempre minore con il decrescere della funzione obiettivo.

7.3.3 Risultati

I risultati in modalità validazione una volta raggiunto questo livello di affinamento dei dati sono stati ricavati in un tempo variabile dai 30 minuti all'ora e mezza e quindi sufficientemente basso da poter valutare differenti configurazioni dei dati in input.

Dati i tempi ristretti e la mancanza di soluzioni precedenti salvate nel database l'ottimizzazione viene lanciata solamente in modalità ottimizzazione globale e dopo una decina di ore di studio del problema inizia un lungo processo di convergenza alla soluzione matematicamente ottima partendo da una prima soluzione con costo della funzione obiettivo di circa 80.000. Dopo qualche giorno di calcolo arriva ad una soluzione con costo di circa 20.000 che viene utilizzata come base dell'orario del prossimo anno[10], al netto di qualche modifica manuale.

[9] A tali vincoli è stata associata una penalità pari a LV_5.

[10] Non è stato possibile utilizzare una soluzione con costo inferiore poichè si sono saturati i 192 GB di RAM del nodo di calcolo.

7.3.4 Esportazione

Quest'ultima fase prevede l'estrazione della soluzione salvata in fase di solving nel database in un formato facilmente interpretabile dall'utente. Sono previsti 3 differenti files che rappresentano l'orario della soluzione:

- Slot orario: file csv contente una rappresentazione sintetica di tutti gli Slot allocati dell'orario con tutte le informazioni rilevanti di supporto come mostrato in Figura 7.4, l'elenco completo dei campi presenti corrisponde a: "PianoAllocazione", "idSlot", "nStudentiAssegnati", "tipoLez", "numSlotConsecutivi", "ID_INC", "Insegnamento", "Titolare", "Giorno", "Ora", "tipoLocale", "tipoErogazione", "capienzaAula", "squadra", "preseElettriche", "Note", "Docenti", "Orientamenti".

PianoAllocazione	idSlot	nStudentiAssegnati	tipoLez	numSlotConsecutivi	ID_INC	Insegnamento
test_relaxed3_opt_20397	260170_slot1	160	L	2	260170	Data management and visualization
test_relaxed3_opt_20397	261048_slot2	190	L	1	261048	Architetture dei sistemi di elaborazione alf:1
test_relaxed3_opt_20397	260444_slot2	80	L	2	260444	Cinema immersivo
test_relaxed3_opt_20397	259944_slot_lab2	125	EL	1	259944	Data Science e Tecnologie per le Basi di Dati

Figura 7.4: File csv Slot allocati.

- Matrice allocazione Insegnamenti: file excel rappresentante una tabella in cui viene indicato per ogni Insegnamento gli slot orari allocati ed il tipo di Locale associato ad essi come in Figura 7.5.

Apiletti Daniele	Big data: architectures and data an	259691						X		
Rizzo Giuseppe	Applied data science project	260184			X			ACSL	ACSLAB	
Lazarescu Mihai Teodor	Applied electronics	261173								
Maffiodo Daniela	Applied mechanics and machine de	259399		X						
Malan Stefano Alberto	Automotive control systems	261627						LAIB	LAIB	

Figura 7.5: Matrice allocazione Insegnamenti negli slot orari.

- Statistiche circa l'occupazione dei Locali nei vari slot orari della giornata come nell'Esempio 17.

  Esempio 17.
  Lun:
  8.30-10.00:
  nSlot totali allocati: 20

nSlot in Aula con capienza Piccola: 7
nSlot in Aula con capienza Media: 3
nSlot in Aula con capienza MedioGrande: 3
nSlot in Aula con capienza Grande: 3
nSlot in Aula con capienza Piccola e necessità prese elettriche: 1
nSlot in Aula con capienza Media e necessità prese elettriche: 0
nSlot in Aula con capienza MedioGrande e necessità prese elettriche: 0
nSlot in Aula con capienza Grande e necessità prese elettriche: 0
nSlot in locale di tipo Aula: 0
nSlot in locale di tipo Laboratorio: 0
nSlot in locale di tipo LABINF: 1
nSlot in locale di tipo LAIB: 0
nSlot in locale di tipo LADISPE: 1
nSlot in locale di tipo LED: 2
nSlot in locale di tipo LED1: 0
nSlot in locale di tipo LED2: 0
nSlot in locale di tipo ACSLAB: 0
nSlot in locale di tipo Aula 5T: 0
nSlot in locale di tipo Aula attrezzata CA2: 4
nSlot in locale di tipo Aula attrezzata CA: 11
nSlot in locale di tipo Aula TableBox: 0
nSlot in locale di tipo Aula WallBox: 1

Capitolo 8

Conclusioni

La pianificazione dell'orario si è rivelata un problema complesso da affrontare per la varietà di vincoli presenti. Si sono dovuti gestire i vincoli strutturali, quali aule e laboratori disponibili in quantità limitata e con capienze note, le dipendenze date dal piano di studio, come ad esempio l'impossibilità di sovrapporre insegnamenti obbligatori per un dato orientamento, i vincoli dettati dalla disponibilità dei docenti che possono essere coinvolti in più insegnamenti, anche esterni ai collegi ICM ed ETF o dovuti ad altri impegni istituzionali. Si è infine tenuto conto delle necessità degli studenti di non avere troppe lezioni consecutive e di avere un limite giornaliero di lezioni definito.

I primi passi per affrontare il problema sono stati l'indivuazione delle due classi di vincoli hard e soft constraint ed uno studio della letteratura informatica per scoprire i modelli di ottimizzazione e le tecnologie allo stato dell'arte andando ad individuare i due problemi presi in esame nello svolgimento della Tesi: MaxSMT e ILP.

È seguita un'analisi dei requisiti conclusa dalla progettazione di alto livello di due modelli UML: il primo in grado di descrivere globalmente il problema ed il secondo specifico dei TemplateOrario, la principale sorgente di dati per il software. Si sono inoltre generati gli schemi di validazione di tutte le sorgenti di dato del software.

Durante la modellazione di basso livello si sono implementati il Modello a Matrice ed il Modello a Slot andando ad evidenziare prima in ambiente matematico e successivamente in ambiente di sviluppo i pregi e i difetti di

entrambi. Dalle conclusioni tratte sui modelli implementati si è scelto il Modello a Slot implementato tramite il software CPLEX.

La successiva fase di testing e validazione ha racchiuso molte difficoltà dovute alla reperibilità e alla correttezza dei dati, sia nel momento in cui erano necessari dati per testare la bontà del software che successivamente per ottenere l'allocazione vera e propria dell'orario. Tra le soluzioni adottate in questa fase si menzionano i vari script per un caricamento semi automatico dei dati sorgente, partendo da tutte le sorgenti, locali e remote, menzionate nel Capitolo 7 e l'applicazione web per una rapida visualizzazione dell'orario mirata a mostrare eventuali violazioni.

La conclusione della Tesi sono state la ricerca e successiva correzione delle incongruenze nei dati sorgente, mediante l'ausilio di log creati appositamente, e la successiva estrazione di un piano di allocazione dell'orario dai risultati del software in esecuzione in modalità ottimizzazione.

I miglioramenti futuri potrebbero includere un nuovo modello di basso livello, in quando il Modello a Slot è stato in grado di modellare correttamente tutti i vincoli emersi dai requisiti ad eccezione dell'allocazione fisica delle differenti aule ai diversi slot di lezione al fine di minimizzare i tempi di spostamento al cambio d'ora. Altri miglioramenti possono riguardare il processo di raccolta dei dati sorgente del problema rendendolo automatizzato ed in grado di garantire dati esaustivi e corretti senza interventi manuali o quasi; discorso valido anche per l'esportazione dei dati, seppure in questo ambito degli script di estrazione dei dati da database siano già presenti. Infine per una maggior immediatezza d'uso la creazione di un'interfaccia grafica in grado di supportare l'utente in tutte le fasi di utilizzo del software potrebbe essere un ulteriore evolutiva del software.

Appendice A

Richiami al codice sviluppato nel Modello a Slot

Questa sezione comprende una raccolta di tutte le equazioni definite all'interno del software per descrivere il modello come un problema di programmazione lineare trattabile da CPLEX.

- Gli slot devono essere assegnati in fasce orarie e giorni validi da (5.39):

```
# for the day
self.X_d = self.model.integer_var_list(self.AUX.get_nSlotId(), 0,
    self.AUX.get_NUM_DAY()-1, "X_d")
# for the hour
self.X_h = self.model.integer_var_list(self.AUX.get_nSlotId(), 0,
    self.AUX.NUM_SLOT_PER_DAY-1, "X_h")

# serve settare almeno un vincolo per ogni variabile del modello,
    altrimenti non viene modellata
eqs.append(self.X_h[slotId] >= 0)

if self.AUX.pianoAllocazione[slotId].numSlot > 1:
    # se lo slot è di un solo blocco non serve aggiungere nessun
    constraint
    eqs.append(self.X_h[slotId] <= self.AUX.NUM_SLOT_PER_DAY - self.
    AUX.pianoAllocazione[slotId].numSlot)
```

- Al Sabato sono disponibili meno fasce orarie da (5.40):

```
# il sabato ho meno slot allocabili
eqs.append(if_then(self.X_d[slotId] == self.AUX.get_NUM_DAY()-1,
   self.X_h[slotId]+self.AUX.pianoAllocazione[slotId].numSlot <= self
    .PARAM.nSlotSabato))
```

- Slot consecutivi come descritto in (5.41)

```
for slotId_i in self.AUX.list_slotInInsegnamento[idInsegnamento]:
    for slotId_j in self.AUX.list_slotInInsegnamento[idInsegnamento]:
        if slotId_i > slotId_j:
            # se gli Slot non sono di tipo EL
            if self.AUX.pianoAllocazione[slotId_i].tipoLezione ==
    TipoLezione.EsercitazioneLaboratorio or self.AUX.pianoAllocazione
    [slotId_j].tipoLezione == TipoLezione.EsercitazioneLaboratorio:
                continue

            # se non c'è nessun Docente in comune tra i 2 slot (
    sebbene i due Slot appartengano allo stesso Insegnamento) i due
    Slot
            # non dovranno essere sovrapposti (vedi
    slotInsegnamentoNellaStessaGiornataNonSovrapposti()), ma non è
    necessario
            # che siano consecutivi
            intersez:bool = False
            for doc_i in self.AUX.list_docentiInSlot[slotId_i]:
                if doc_i in self.AUX.list_docentiInSlot[slotId_j]:
                    intersez = True
            if not intersez:
                continue

            eq = if_then(self.X_d[slotId_i] == self.X_d[slotId_j],
                logical_or(self.X_h[slotId_i] == self.X_h[slotId_j] +
     self.AUX.pianoAllocazione[slotId_j].numSlot,
                self.X_h[slotId_j] == self.X_h[slotId_i] + self.AUX.
    pianoAllocazione[slotId_i].numSlot))
```

- Docente allocato in un solo slot orario alla volta da (5.42):

```
for slotId_i in self.AUX.list_slotInDocente[docId]:
    for slotId_j in self.AUX.list_slotInDocente[docId]:
        # se i due Slot appartengono allo stesso docente -> non possono essere allocati contemporaneamente

        # se i due Slot si riferiscono entrambi ad Insegnamenti sovrapponibili E APPARTENGONO ad INSEGNAMENTI DIFFERENTI
        if docId in self.map_docentiConInsegnamentiSovrapponibili.keys():
            if self.AUX.pianoAllocazione[slotId_i].idInsegnamento != self.AUX.pianoAllocazione[slotId_j].idInsegnamento:

                ID_INC_i:int = int(self.AUX.list_Insegnamenti[self.AUX.pianoAllocazione[slotId_i].idInsegnamento].ID_INC)
                ID_INC_j:int = int(self.AUX.list_Insegnamenti[self.AUX.pianoAllocazione[slotId_j].idInsegnamento].ID_INC)

                if ID_INC_i in self.map_docentiConInsegnamentiSovrapponibili[docId] and ID_INC_j in self.map_docentiConInsegnamentiSovrapponibili[docId]:
                    continue

        if slotId_j > slotId_i: # as matrice triangolare (gli slotId sono ordinati)
            eq = if_then(X_d[slotId_i] == X_d[slotId_j],
                logical_or(X_h[slotId_i] >= X_h[slotId_j] + self.AUX.pianoAllocazione[slotId_j].numSlot,
                X_h[slotId_j] >= X_h[slotId_i] + self.AUX.pianoAllocazione[slotId_i].numSlot))
```

- L'ultimo slot orario di un giorno ed il primo del successivo non possono essere allocati per lo stesso Insegnamento come da (5.43):

```
for slotId_i in self.AUX.list_slotInInsegnamento[idInsegnamento]:
    for slotId_j in self.AUX.list_slotInInsegnamento[idInsegnamento]:

        if self.AUX.pianoAllocazione[slotId_i].tipoLezione == TipoLezione.EsercitazioneLaboratorio or self.AUX.pianoAllocazione[slotId_j].tipoLezione == TipoLezione.EsercitazioneLaboratorio:
```

```
            continue

        if self.AUX.pianoAllocazione[slotId_i].squadra != Squadra.
    NoSquadra or self.AUX.pianoAllocazione[slotId_j].squadra !=
    Squadra.NoSquadra:
            continue

        if slotId_i > slotId_j:

            eq1 = if_then(
                logical_and(self.X_d[slotId_i]+1 == self.X_d[slotId_j
    ],
                self.X_h[slotId_j]+self.AUX.pianoAllocazione[slotId_j
    ].numSlot == 7),
                self.X_h[slotId_i] > 0)

            eq2 = if_then(
                logical_and(self.X_d[slotId_j]+1 == self.X_d[slotId_i
    ],
                self.X_h[slotId_i]+self.AUX.pianoAllocazione[slotId_i
    ].numSlot == 7),
                self.X_h[slotId_j] > 0)
```

- Slot di uno stesso Insegnamento non sovrapposti salvo casi particoli come specificato nell'equazione (5.44)

```
for slotId_i in self.AUX.list_slotInInsegnamento[idInsegnamento]:
    for slotId_j in self.AUX.list_slotInInsegnamento[idInsegnamento]:
        if slotId_i > slotId_j:
            # se i due Slot hanno Docenti in comune => sono
    sicuramente consecutivi da self.
    slotInsegnamentoNellaStessaGiornataConsecutivi()
            isIntersezioneDocenti:bool = False
            for doc_i in self.AUX.list_docentiInSlot[slotId_i]:
                if doc_i in self.AUX.list_docentiInSlot[slotId_j]:
                    isIntersezioneDocenti = True

            # se anche solo uno degli Slot non è a squadre => non
    posso allocare in parallelo
            isSquadre:bool = (self.AUX.pianoAllocazione[slotId_i].
    squadra != Squadra.NoSquadra and self.AUX.pianoAllocazione[
    slotId_j].squadra != Squadra.NoSquadra)

```

```
            # se le squadre sono le stesse => non posso allocare in parallelo
            isSameSquadra:bool = (self.AUX.pianoAllocazione[slotId_i].squadra == self.AUX.pianoAllocazione[slotId_j].squadra)

            # Slot della stessa squadra o che non fanno riferimento a squadre non devono essere consecutivi
            if isSquadre and not isSameSquadra:
                continue
            if not isIntersezioneDocenti:
                eq = if_then(self.X_d[slotId_i] == self.X_d[slotId_j],
                    logical_or(self.X_h[slotId_i] >= self.X_h[slotId_j] + self.AUX.pianoAllocazione[slotId_j].numSlot,
                    self.X_h[slotId_j] >= self.X_h[slotId_i] + self.AUX.pianoAllocazione[slotId_i].numSlot))
```

- Slot scelti di un Insegnamento da (5.45)[1]:

```
if self.AUX.pianoAllocazione[slotId].tipoSlot == TipoSlot.SlotScelto:
    eqs.append(self.X_d[slotId] == self.AUX.pianoAllocazione[slotId].daySlotAllocato)
    eqs.append(self.X_h[slotId] == self.AUX.pianoAllocazione[slotId].hourSlotAllocato)
elif self.AUX.pianoAllocazione[slotId].tipoSlot == TipoSlot.SlotScelto_fasciaOraria:
    eqs.append(self.X_h[slotId] == self.AUX.pianoAllocazione[slotId].hourSlotAllocato)
elif self.AUX.pianoAllocazione[slotId].tipoSlot == TipoSlot.SlotScelto_giorno:
    eqs.append(self.X_d[slotId] == self.AUX.pianoAllocazione[slotId].daySlotAllocato)
```

- Slot di Insegnamenti appartenenti allo stesso Orientamento non possono sovrapporsi come soft o hard constraint come da (5.46)[2]:

[1]Viene introdotta la possibilità di preallocare anche solo il giorno o la fascia oraria dello Slot.

[2]Viene omessa la logica di generazione dei vincoli

```
# Hard Constraint
eq = if_then(X_d[rule.slotId_i] == X_d[rule.slotId_j],
        logical_or(X_h[rule.slotId_i] >= X_h[rule.slotId_j] + self.AUX.pianoAllocazione[rule.slotId_j].numSlot,
            X_h[rule.slotId_j] >= X_h[rule.slotId_i] + self.AUX.pianoAllocazione[rule.slotId_i].numSlot))

# Soft Constraint
 eq = if_then(
        logical_and(
            logical_and(X_d[rule.slotId_i] == X_d[rule.slotId_j],
                logical_not(X_h[rule.slotId_i] >= X_h[rule.slotId_j] + self.AUX.pianoAllocazione[rule.slotId_j].numSlot)
            ), logical_not(X_h[rule.slotId_j] >= X_h[rule.slotId_i] + self.AUX.pianoAllocazione[rule.slotId_i].numSlot)
    ),
    X_penalties[index] == rule.valPenalita)
```

- Preferenza su quando allocare o non allocare gli Slot di un Insegnamento come in (5.49) e (5.50)

```
# Hard Constraint -> per indicare indisponibilità Insegnamento
eq = if_then(X_d[rule.slotId] == getIntFromDay(rule.day),
        logical_or( # inizia dopo o finisce prima
             X_h[rule.slotId] > getIntFromFasciaOraria(rule.fasciaOraria), X_h[rule.slotId] + self.AUX.pianoAllocazione[rule.slotId].numSlot <= getIntFromFasciaOraria(rule.fasciaOraria)
        ))

# Soft Constraint
# se lo Slot è allocato quel giorno e NON inizia dopo O NON finisce prima
eq = if_then(logical_and(
        X_d[rule.slotId] == getIntFromDay(rule.day),
            logical_not(logical_or(
                X_h[rule.slotId] > getIntFromFasciaOraria(rule.fasciaOraria),
                X_h[rule.slotId] + self.AUX.pianoAllocazione[rule.slotId].numSlot <= getIntFromFasciaOraria(rule.fasciaOraria)
            ))),
```

```
            X_penalties[index] == getIntFromLV_Penalties(rule.
    penalita, TipoPenalty.Orientamento))
eq1 = if_then(logical_not(logical_and(
        X_d[rule.slotId] == getIntFromDay(rule.day),
            logical_not(logical_or(
                X_h[rule.slotId] > getIntFromFasciaOraria(rule.
    fasciaOraria),
                X_h[rule.slotId] + self.AUX.pianoAllocazione[rule.
    slotId].numSlot <= getIntFromFasciaOraria(rule.fasciaOraria)
            )))),
            X_penalties[index] == 0)
```

- Allineamento X_Orient come specificato da (5.51)[3]:

```
if self.AUX.list_slotIdInOrientamento_periodoDidattico[orientId][
    index_slot] in [PeriodoDidattico.PrimoAnno_PrimoSemestre,
    PeriodoDidattico.PrimoAnno_SecondoSemestre]:
    eq = if_then(logical_and(X_d[slotId] == day, X_h[slotId] == hour)
    , self.X_orient[self.helper_getOffsetOrient(orientId,1)+day*self.
    AUX.NUM_SLOT_PER_DAY+hour+nSlot_durata] == 1)
```

- Limite di 6 slot orari giornalieri come specificato da (5.52):

```
eqMax = model.sum(self.X_orient[self.helper_getOffsetOrient(orientId
    ,3)+day*self.AUX.NUM_SLOT_PER_DAY+i] == 1 for i in range(self.AUX.
    NUM_SLOT_PER_DAY)) <= 6
```

- Limite di 5 slot orari consecutivi come specificato da (5.53):

```
# se ho 6 slot allocati => il primo e l'ultimo della giornata devono
    essere entrambi allocati
# per avere l'ora buca non agli estremi della giornata e quindi avere
     max 5 slot consecutivi
```

[3] *self.helper_getOffsetOrient(..) ritorna il corretto pStud associato all'equazione corrente.*

```
eqCons = if_then(model.sum(self.X_orient[self.helper_getOffsetOrient(
    orientId,3)+day*self.AUX.NUM_SLOT_PER_DAY+i] == 1 for i in range(
    self.AUX.NUM_SLOT_PER_DAY)) == 6,
     logical_and(self.X_orient[self.helper_getOffsetOrient(orientId,3)
    +day*self.AUX.NUM_SLOT_PER_DAY] == 1,
         self.X_orient[self.helper_getOffsetOrient(orientId,3)+day*
    self.AUX.NUM_SLOT_PER_DAY+self.AUX.NUM_SLOT_PER_DAY-1] == 1))
```

- Penalità se ci sono 4 slot orari consecutivi come specificato da (5.54)[4]:

```
listOptEq = list()
for hStart in range(self.AUX.NUM_SLOT_PER_DAY-3):
    eq_if = model.sum(
        self.X_orient[self.helper_getOffsetOrient(orientId,year)+day*
    self.AUX.NUM_SLOT_PER_DAY + hStart+i] == 1
            for i in range(4)) == 4
    listOptEq.append(eq_if)

# pseudocode
# eq = if_then(logical_or(listOptEq[0], ...), X_penaltiesStud[pStud]
    == LV_Penalties.LV_1)
self.listRuleStudente.append(RuleStudente(rOr(listOptEq),
    LV_Penalties.LV_1,
    self.helper_getOffsetPenalita(orientId, year, day),
    orientId, year, day, TipoRuleStudente.SlotConsecutivi,
    self.helper_getOffsetOrient(orientId,year)+day*self.AUX.
    NUM_SLOT_PER_DAY))
```

- Penalità se c'è un intervallo da due slot orari come specificato da (5.55):

```
listOptEq = list()
eq_if = logical_and(model.sum(
    self.X_orient[self.helper_getOffsetOrient(orientId,year)+day*self
    .AUX.NUM_SLOT_PER_DAY + i]
       for i in range(buco))
    == model.sum(
```

[4] *rOr(eqs)* è una funzione che mette in or logico tra loro la lista di equazioni passata come parametro.

```
        self.X_orient[self.helper_getOffsetOrient(orientId,year)+day*
    self.AUX.NUM_SLOT_PER_DAY + i]
            for i in range(buco+2))
    ,
        model.sum(
            self.X_orient[self.helper_getOffsetOrient(orientId,year)+
    day*self.AUX.NUM_SLOT_PER_DAY + i]
                for i in range(buco))
        != model.sum(
            self.X_orient[self.helper_getOffsetOrient(orientId,year)+
    day*self.AUX.NUM_SLOT_PER_DAY + j]
                for j in range(buco+3)))
                        # no self.AUX.NUM_SLOT_PER_DAY perchè
    altrimenti saprei solo se ho un buco di ALMENO 3 slot
listOptEq.append(eq_if)

# pseudocode
# eq = if_then(logical_or(listOptEq[0], ...), X_penaltiesStud[pStud]
    == LV_Penalties.LV_1)
self.listRuleStudente.append(RuleStudente(rOr(listOptEq),
    LV_Penalties.LV_1,
    nVarSlotConsecutivi + self.helper_getOffsetPenalita(orientId,
    year, day),
    orientId, year, day, TipoRuleStudente._1BucoOrario,
    self.helper_getOffsetOrient(orientId,year)+day*self.AUX.
    NUM_SLOT_PER_DAY))
```

- Penalità se ci sono 2 intervalli lo stesso giorno come specificato da (5.56):

```
for buco1 in range(1,self.AUX.NUM_SLOT_PER_DAY-1-2):
    for buco2 in range(buco1+2, self.AUX.NUM_SLOT_PER_DAY-1):
        eq_if = logical_and(
            logical_and(
                logical_and(
                    model.sum(
                        self.X_orient[self.helper_getOffsetOrient(
    orientId,year)+day*self.AUX.NUM_SLOT_PER_DAY + i]
                            for i in range(buco1))
                    == model.sum(
                        self.X_orient[self.helper_getOffsetOrient(
    orientId,year)+day*self.AUX.NUM_SLOT_PER_DAY + i]
                            for i in range(buco1+1))
                ,
```

```
                    model.sum(
                        self.X_orient[self.helper_getOffsetOrient(
orientId,year)+day*self.AUX.NUM_SLOT_PER_DAY + i]
                            for i in range(buco1))
                    != model.sum(
                        self.X_orient[self.helper_getOffsetOrient(
orientId,year)+day*self.AUX.NUM_SLOT_PER_DAY + i]
                            for i in range(buco2))
                    )
                ,
                    model.sum(
                        self.X_orient[self.helper_getOffsetOrient(
orientId,year)+day*self.AUX.NUM_SLOT_PER_DAY + i]
                            for i in range(buco2))
                    == model.sum(
                        self.X_orient[self.helper_getOffsetOrient(
orientId,year)+day*self.AUX.NUM_SLOT_PER_DAY + i]
                            for i in range(buco2+1))
                    )
                ,
                    model.sum(
                        self.X_orient[self.helper_getOffsetOrient(
orientId,year)+day*self.AUX.NUM_SLOT_PER_DAY + i]
                            for i in range(buco2))
                    !=  model.sum(
                        self.X_orient[self.helper_getOffsetOrient(
orientId,year)+day*self.AUX.NUM_SLOT_PER_DAY + i]
                            for i in range(self.AUX.NUM_SLOT_PER_DAY)
)
                    )
        # pseudocode
        # eq = if_then(eq_if, X_penaltiesStud[pStud] == LV_Penalties.
LV_6)
        self.listRuleStudente.append(RuleStudente(eq_if, LV_Penalties
.LV_6,
            nVarSlotConsecutivi + nVar1Buco + self.
helper_getOffsetPenalita(orientId, year, day),
            orientId, year, day, TipoRuleStudente._2BuchiOrario,
            self.helper_getOffsetOrient(orientId,year)+day*self.AUX.
NUM_SLOT_PER_DAY))
```

- Limite disponibilità Locali da (5.57):

```
eq1 = model.sum((X_d[slotId] == day)*(X_h[slotId] <= h)*((X_h[slotId]+self.AUX.pianoAllocazione[slotId].numSlot > h))
    for slotId in self.listSlot_ACSLAB) <= self.getLimit(day, h, TipoLocale.ACSLAB)
```

- Limite slot orari allocati per un Docente da (5.58)

```
eqs.append(model.sum([(X_d[slotId] == day) * self.AUX.pianoAllocazione[slotId].numSlot
 for slotId in self.AUX.list_slotInDocente[docId]]) <= self.map_limitHourDocentePerDay[docId])
```

- Distanza minima tra due Slot associati ad un Docente da (5.59) e (5.60):

```
# Hard Constraint
eq = if_then(X_d[rule.slotId_i] == X_d[rule.slotId_j],
    logical_or(
        X_h[rule.slotId_i] >= X_h[rule.slotId_j] + self.AUX.pianoAllocazione[rule.slotId_j].numSlot + rule.dist,
        X_h[rule.slotId_j] >= X_h[rule.slotId_i] + self.AUX.pianoAllocazione[rule.slotId_i].numSlot + rule.dist
    ))

# Soft Constraint
eq = if_then(logical_and(
        X_d[rule.slotId_i] == X_d[rule.slotId_j],
        logical_not(logical_or(
            X_h[rule.slotId_i] >= X_h[rule.slotId_j] + self.AUX.pianoAllocazione[rule.slotId_j].numSlot + rule.dist,
            X_h[rule.slotId_j] >= X_h[rule.slotId_i] + self.AUX.pianoAllocazione[rule.slotId_i].numSlot + rule.dist
        ))
    ),
        X_penaltiesDoc[index] == getIntFromLV_Penalties(rule.penalita, TipoPenalty.Docente)
    )
eq1 = if_then(logical_not(logical_and(
        X_d[rule.slotId_i] == X_d[rule.slotId_j],
        logical_not(logical_or(
```

```
            X_h[rule.slotId_i] >= X_h[rule.slotId_j] + self.AUX.
    pianoAllocazione[rule.slotId_j].numSlot + rule.dist,
            X_h[rule.slotId_j] >= X_h[rule.slotId_i] + self.AUX.
    pianoAllocazione[rule.slotId_i].numSlot + rule.dist
        ))
    )),
        X_penaltiesDoc[index] == 0
    )
```

- Distanza massima tra due Slot associati ad un Docente da (5.61) e (5.62):

```
# Hard Constraint
eq = if_then(logical_and(
        X_d[rule.slotId_i] == X_d[rule.slotId_j],
        X_h[rule.slotId_i] > X_h[rule.slotId_j]
    ),
        X_h[rule.slotId_i] <= X_h[rule.slotId_j] + self.AUX.
    pianoAllocazione[rule.slotId_j].numSlot + rule.dist
    )
eq1 = if_then(logical_and(
        X_d[rule.slotId_i] == X_d[rule.slotId_j],
        X_h[rule.slotId_j] > X_h[rule.slotId_i]
    ),
        X_h[rule.slotId_j] <= X_h[rule.slotId_i] + self.AUX.
    pianoAllocazione[rule.slotId_i].numSlot + rule.dist
    )

# Soft Constraint
eq = if_then(logical_and(
    X_d[rule.slotId_i] == X_d[rule.slotId_j],
    logical_or(
        logical_and(
           X_h[rule.slotId_i] > X_h[rule.slotId_j],
           X_h[rule.slotId_i] > X_h[rule.slotId_j] + self.AUX.
    pianoAllocazione[rule.slotId_j].numSlot + rule.dist
        ), logical_and(
            X_h[rule.slotId_j] > X_h[rule.slotId_i],
            X_h[rule.slotId_j] > X_h[rule.slotId_i] + self.AUX.
    pianoAllocazione[rule.slotId_i].numSlot + rule.dist
        )
    )),
```

```
    X_penaltiesDoc[index] == getIntFromLV_Penalties(rule.penalita,
    TipoPenalty.Docente))
eq1 = if_then(logical_not(logical_and(
    X_d[rule.slotId_i] == X_d[rule.slotId_j],
    logical_or(
        logical_and(
           X_h[rule.slotId_i] > X_h[rule.slotId_j],
           X_h[rule.slotId_i] > X_h[rule.slotId_j] + self.AUX.
    pianoAllocazione[rule.slotId_j].numSlot + rule.dist
        ), logical_and(
            X_h[rule.slotId_j] > X_h[rule.slotId_i],
            X_h[rule.slotId_j] > X_h[rule.slotId_i] + self.AUX.
    pianoAllocazione[rule.slotId_i].numSlot + rule.dist
        )
    ))),
    X_penaltiesDoc[index] == 0)
```

- Slot di Insegnamenti differenti associati allo stesso Docente non dovrebbero essere allocati lo stesso giorno da (5.63) e (5.64):

```
# Hard Constraint
eq = X_d[rule.slotId_i] != X_d[rule.slotId_j]

# Soft Constraint
eq = if_then(
    X_d[rule.slotId_i] == X_d[rule.slotId_j],
    X_penaltiesDoc[index] == getIntFromLV_Penalties(rule.penalita,
    TipoPenalty.Docente)
)
eq1 = if_then(
    X_d[rule.slotId_i] != X_d[rule.slotId_j],
    X_penaltiesDoc[index] == 0
)
```

- Slot s1 almeno un giono dopo un altro Slot s2 da (5.65) e (5.66):

```
# Hard Constraint
eq = X_d[self.slotId_i] > X_d[self.slotId_j]

# Soft Constraint
```

```
eq = if_then(X_d[self.slotId_i] > X_d[self.slotId_j],
    X_penaltiesDoc[index] == (-1)*getIntFromLV_Penalties(vincolo.
    penalita, TipoPenalty.Docente))
eq1 = if_then(logical_not(X_d[self.slotId_i] > X_d[self.slotId_j]),
    X_penaltiesDoc[index] == (-1)*getIntFromLV_Penalties(vincolo.
    penalita, TipoPenalty.Docente))
```

- Slot s1 in un giorno diverso da un altro Slot s2 da (5.67) e (5.68):

```
# Hard Constraint
eq = X_d[self.slotId_i] != X_d[self.slotId_j]

# Soft Constraint
eq = if_then(X_d[self.slotId_i] != X_d[self.slotId_j],
    X_penaltiesDoc[index] == (-1)*getIntFromLV_Penalties(vincolo.
    penalita, TipoPenalty.Docente))
eq1 = if_then(logical_not(X_d[self.slotId_i] != X_d[self.slotId_j]),
    X_penaltiesDoc[index] == (-1)*getIntFromLV_Penalties(vincolo.
    penalita, TipoPenalty.Docente))
```

- Slot s1 stesso giorno di un altro Slot s2 da (5.69) e (5.70):

```
# Hard Constraint
eq = X_d[self.slotId_i] == X_d[self.slotId_j]

# Soft Constraint
eq = if_then(X_d[self.slotId_i] == X_d[self.slotId_j],
    X_penaltiesDoc[index] == (-1)*getIntFromLV_Penalties(vincolo.
    penalita, TipoPenalty.Docente))
eq1 = if_then(logical_not(X_d[self.slotId_i] == X_d[self.slotId_j]),
    X_penaltiesDoc[index] == (-1)*getIntFromLV_Penalties(vincolo.
    penalita, TipoPenalty.Docente))
```

- Slot s1 parallelo ad un altro Slot s2 da (5.71) e (5.72):

```
# Hard Constraint
eq = logical_and(X_d[self.slotId_i] == X_d[self.slotId_j], X_h[self.
    slotId_i] == X_h[self.slotId_j])
```

```
3
4 # Soft Constraint
5 eq = if_then(logical_and(X_d[self.slotId_i] == X_d[self.slotId_j],
      X_h[self.slotId_i] == X_h[self.slotId_j]),
6      X_penaltiesDoc[index] == (-1)*getIntFromLV_Penalties(vincolo.
      penalita, TipoPenalty.Docente))
7 eq1 = if_then(logical_not(logical_and(X_d[self.slotId_i] == X_d[self.
      slotId_j], X_h[self.slotId_i] == X_h[self.slotId_j])),
8      X_penaltiesDoc[index] == (-1)*getIntFromLV_Penalties(vincolo.
      penalita, TipoPenalty.Docente))
```

- Slot s1 subito dopo un altro Slot s2 da (5.73) e (5.74):

```
1 # Hard Constraint
2 eq = logical_and(X_d[self.slotId_i] == X_d[self.slotId_j], X_h[self.
      slotId_i] == X_h[self.slotId_j] + self.numSlot_j)
3
4 # Soft Constraint
5 eq = if_then(logical_and(X_d[self.slotId_i] == X_d[self.slotId_j],
      X_h[self.slotId_i] == X_h[self.slotId_j] + self.numSlot_j),
6      X_penaltiesDoc[index] == (-1)*getIntFromLV_Penalties(vincolo.
      penalita, TipoPenalty.Docente))
7 eq1 = if_then(logical_not(logical_and(X_d[self.slotId_i] == X_d[self.
      slotId_j], X_h[self.slotId_i] == X_h[self.slotId_j] + self.
      numSlot_j)),
8      X_penaltiesDoc[index] == (-1)*getIntFromLV_Penalties(vincolo.
      penalita, TipoPenalty.Docente))
```

Bibliografia

[1] Yamada Takeshi and Nakano Ryohei. *Job-shop scheduling*. Tech. rep. 1997. URL: https://www.math.unipd.it/~luigi/courses/rodid/m01.simplex.pdf.

[2] Sabino Roselli, Kristofer Bengtsson, and Knut Åkesson. «SMT Solvers for Job-Shop Scheduling Problems: Models Comparison and Performance Evaluation». In: Aug. 2018, pp. 547–552. DOI: 10.1109/COASE.2018.8560344.

[3] Wikipedia. *Conjunctive normal form*. 2022. URL: https://en.wikipedia.org/wiki/Conjunctive_normal_form.

[4] Leonardo de Moura and Nikolaj Bjørner. *Z3 - a Tutorial*. Tech. rep. 2019. URL: https://www.cs.colostate.edu/~cs440/spring19/slides/z3-tutorial.pdf.

[5] Nikolaj Bjørner, Leonardo de Moura, Lev Nachmanson, and Christoph Wintersteiger. *Programming Z3*. 2022. URL: https://theory.stanford.edu/~nikolaj/programmingz3.html.

[6] Nikolaj Bjørner, Leonardo de Moura, Lev Nachmanson, and Christoph M. Wintersteiger. «Programming Z3». In: *Engineering Trustworthy Software Systems: 4th International School, SETSS 2018, Chongqing, China, April 7–12, 2018, Tutorial Lectures*. Ed. by Jonathan P. Bowen, Zhiming Liu, and Zili Zhang. Cham: Springer International Publishing, 2019, pp. 148–201. ISBN: 978-3-030-17601-3. DOI: 10.1007/978-3-030-17601-3_4. URL: https://doi.org/10.1007/978-3-030-17601-3_4.

[7] Luigi De Giovanni and Giacomo Zambelli. *Introduzione al metodo del simplesso*. Tech. rep. 2022. URL: https://www.math.unipd.it/~luigi/courses/rodid/m01.simplex.pdf.

[8] IBM. *Tutorial: Linear Programming, (CPLEX Part 1)*. 2022. URL: `https://ibmdecisionoptimization.github.io/tutorials/html/Linear_Programming.html`.

[9] IBM. *IBM ILOG CPLEX Optimization Studio 20.1.0*. 2020. URL: `https://ibmdecisionoptimization.github.io/tutorials/html/Linear_Programming.html`.