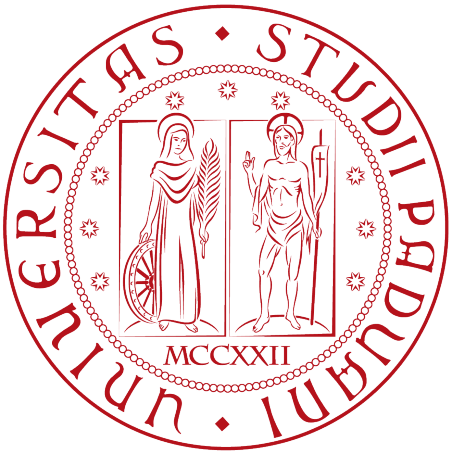
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Analisi e Implementazione di Algoritmi per problemi di Scheduling

Tesi di laurea triennale

Relatore

Prof. Luigi De Giovanni

Laureanda

Beatrice Liberi

Anno Accademico 2016-2017

Sommario

Il presente documento descrive il lavoro che ho svolto durante il periodo di stage, della durata di trecento ore, presso l'azienda Trans-Cel Autotrasporti.

Lo scopo dello stage è stato lo studio di modelli per problemi di scheduling e l'implementazione di algoritmi euristici che restituiscano una soluzione ammissibile in tempi sufficientemente brevi per una pianificazione in tempo reale. Il progetto ha fornito a Trans-Cel Autotrasporti uno strumento che potrà essere integrato nei software di pianificazion sviluppati in azienda e da applicare in diversi contesti per i quali sia necessaria la risoluzione di un problema di scheduling.

Gli obiettivi da raggiungere nel corso dello stage erano molteplici: in primo luogo era richiesto di assimilare i concetti di base dei problemi di scheduling, studiare la letteratura scientifica sui modelli e sugli algoritmi euristici più adatti per risolvere gli stessi; in secondo luogo era richiesto di definire formalmente con un modello matematico un problema di scheduling da risolvere, e progettare un algoritmo euristico per trovare una soluzione al problema; terzo ed ultimo obiettivo era l'implementazione dell'algoritmo euristico, integrandolo nelle librerie dell'azienda.

Indice

1 Introduzione **1**
- 1.1 L'azienda . . . 1
- 1.2 L'idea . . . 1
- 1.3 Organizzazione del testo . . . 2

2 Descrizione dello stage **3**
- 2.1 Introduzione al progetto e contestualizzazione . . . 3
- 2.2 Vincoli metodologici, temporali e tecnologici . . . 3
- 2.3 Obiettivi . . . 4
- 2.4 Pianificazione . . . 6
- 2.5 Ambiente di lavoro . . . 9
  - 2.5.1 Metodologia di sviluppo . . . 9
  - 2.5.2 Gestione di progetto . . . 9
  - 2.5.3 Documentazione . . . 10
  - 2.5.4 Ambiente di sviluppo . . . 10
- 2.6 Analisi preventiva dei rischi . . . 12

3 Definizione del problema **13**
- 3.1 Problemi di scheduling . . . 13
- 3.2 Definizione formale degli elementi generali per i problemi di scheduling 14
  - 3.2.1 Dati . . . 14
  - 3.2.2 Vincoli . . . 14
  - 3.2.3 Obiettivo . . . 15
- 3.3 Definizione del problema: scheduling nei casinò . . . 15
  - 3.3.1 Analisi discorsiva . . . 15
- 3.4 Modello per lo scheduling nei casinò . . . 16
  - 3.4.1 Dati . . . 16
  - 3.4.2 Vincoli . . . 17
  - 3.4.3 Obiettivo . . . 18
- 3.5 Algoritmi euristici di risoluzione per problemi di scheduling . . . 18

4 Analisi dei requisiti **23**
- 4.1 Casi d'uso . . . 23
  - 4.1.1 Diagrammi dei casi d'uso . . . 24
  - 4.1.2 UC1: Inserimento dei dati di input . . . 25
  - 4.1.3 UC2: Inserimento dati lavoratori . . . 25
  - 4.1.4 UC3: Inserimento dati postazioni . . . 27
  - 4.1.5 UC4: Inserimento dati turni . . . 28

4.1.6 UC5: Inserimento data . . . 29
4.1.7 UC6: Inserimento numero turni . . . 29
4.1.8 UC7: Visualizzazione dello scheduling in output . . . 29
4.2 Requisiti . . . 30
4.2.1 Requisiti Funzionali . . . 30
4.2.2 Requisiti di Qualità . . . 34
4.2.3 Requisiti di Vincolo . . . 34

5 Progettazione e codifica **35**
5.1 Premessa: architettura e funzionamento del framework . . . 35
5.1.1 Il metodo run() . . . 36
5.2 Progettazione dell'estensione del framework . . . 38
5.3 Progettazione dell'applicazione . . . 39
5.4 Librerie . . . 40
5.4.1 ExcelFormat . . . 40
5.4.2 Boost . . . 40
5.5 Design pattern . . . 41
5.6 Progettazione di dettaglio . . . 41
5.6.1 Gerarchia di Item . . . 42
5.6.2 SchedulingGreedy . . . 45
5.7 Codifica e sofware realizzato . . . 46

6 Verifica e validazione **47**
6.1 Generazione degli input . . . 47
6.1.1 Modello probabilistico . . . 48
6.2 Testing . . . 52
6.2.1 Costruzione dei test . . . 52
6.2.2 Analisi dei risultati . . . 56
6.2.3 Conclusioni . . . 58

7 Conclusioni **59**
7.1 Raggiungimento degli obiettivi . . . 59
7.2 Resoconto dell'analisi dei rischi . . . 60
7.3 Consuntivo finale . . . 61
7.4 Valutazione personale e conoscenze acquisite . . . 63

Glossario **65**

Acronimi **71**

Bibliografia **73**

Elenco delle figure

2.1 Diagramma di Gantt per il piano di lavoro, settimane 1-5 8
2.2 Diagramma di Gantt per il piano di lavoro, settimane 5-9 8

3.1 Diagramma di attività per l'algoritmo greedy di risoluzione per i problemi di scheduling 20
3.2 Diagramma di attività per l'algoritmo greedy di risoluzione per i problemi di scheduling con aggiunta di Dummy Worker 21

4.1 Diagramma dei casi d'uso ad alto livello 24
4.2 UC2: Inserimento dati lavoratori 25
4.3 UC3: Inserimento dati postazioni 27
4.4 UC4: Inserimento dati turni 28

5.1 Architettura di dettaglio di parte del framework aziendale 36
5.2 Architettura generale dell'estensione al framework 38
5.3 Architettura generale dell'applicazione al problema del casinò 39
5.4 Template Method 41
5.5 Gerarchia della classe Item 42
5.6 Gerarchia della classe Greedy 45

6.1 Distribuzioni normali per il numero di postazioni 48
6.2 Distribuzione discreta uniforme per i livelli delle postazioni 49
6.3 Frequenza di estrazione dei giochi per ogni postazione 49
6.4 Distribuzioni normali per il numero di lavoratori 50
6.5 Distribuzioni normali per i livelli dei lavoratori 51
6.6 Probabilità che due lavoratori di livelli 1 (a sinistra) e 8 (a destra) conoscano un gioco (1) o meno (0) 51
6.7 Grafico prodotto dai test: Executions 53
6.8 Grafico prodotto dai test: Tests (livelli) 54
6.9 Grafico prodotto dai test: Tests (giochi) 54
6.10 Grafico prodotto dai test: Tests (giochi per livelli) 55
6.11 Grafici prodotti dai test: Exec n 56

Elenco delle tabelle

2.1 Obiettivi da raggiungere a fine stage 5
2.2 Pianificazione del periodo di stage 7
2.3 Analisi preventiva dei rischi 12

4.1 Tabella dei requisiti funzionali 31
4.2 Tabella dei requisiti di qualità 34
4.3 Tabella dei requisiti di vincolo 34

6.1 Risultati del primo test prestazionale 56
6.2 Risultati del secondo test prestazionale 57
6.3 Risultati del terzo test prestazionale 58

7.1 Soddisfacimento dei requisiti 59
7.2 Attualizzazione dei rischi 60
7.3 Differenza ore consuntivo-preventivo 62

Capitolo 1

Introduzione

In questo capitolo vengono brevemente descritte l'azienda Trans-Cel Autotrasporti presso la quale ho svolto lo stage e l'idea dalla quale è nata la necessità del progetto portato a termine durante lo stesso.
Viene inoltre presentata la suddivisione della tesi per capitoli e vengono introdotte alcune norme tipografiche che verranno utilizzate di seguito.

1.1 L'azienda

Trans-Cel Autotrasporti è un'azienda che si occupa di trasporti su gomma di merci per conto terzi su mezzi pesanti. Con una flotta di venticinque camion che deve compiere carichi e scarichi in tutta l'Italia centro-settentrionale da coordinare in tempo reale, da qualche anno l'azienda ha cominciato a sviluppare, grazie ad un team di informatici e matematici, un sistema per il controllo della flotta stessa e, soprattutto, per la pianificazione di viaggi, carichi e scarichi in modo da ottimizzare sia l'utilizzo dello spazio disponibile sui camion, sia i chilometri percorsi; il tutto vincolato ai tempi concordati di ritiro e consegna.
Trans-Cel Autotrasporti vede questo planning come il primo mattone di un sistema molto più articolato che unirà diversi tipi di servizi per il supporto di tutta la *supply chain*[g], non solo nel campo dei trasporti.

1.2 L'idea

Uno degli ulteriori servizi che Trans-Cel Autotrasporti vuole sviluppare consiste in un software per la gestione della *schedulazione*[g] (o *scheduling*) dei turni di lavoro.
I *problemi di scheduling*[g] ricadono generalmente nella classe *NP-Hard*[g] e risulta quindi particolarmente difficile trovare delle soluzioni ottime, e spesso anche solo soluzioni ammissibili. Inoltre, esistono diversi contesti in cui i turni devono poter essere proposti in tempi molto rapidi, ad esempio per adeguarsi a cambiamenti durante l'orizzonte di pianificazione. Uno degli utilizzi che Trans-Cel Autotrasporti potrebbe fare di questo software, nel particolare, può essere l'organizzazione delle squadre di meccanici.
Lo stage si pone dunque in questo contesto di progettazione di un framework per la risoluzione dei problemi di scheduling.

1.3 Organizzazione del testo

Il Capitolo 2 descrive in dettaglio lo stage. Ne specifica il progetto da svolgere contestualizzandolo nella realtà aziendale e, definendone i requisiti, gli obiettivi da raggiungere e la pianificazione iniziale.

Il Capitolo 3 si approfondisce l'argomento dei problemi di scheduling e si definisce nei dettagli il problema da risolvere durante lo stage.

Il Capitolo 4 consiste nell'analisi dei requisiti svolta per il progetto, approfondita con diagrammi dei casi d'uso.

Il Capitolo 5 presenta la progettazione svolta per il progetto, approfondita con diagrammi *UML*[g], e ne descrive la fase di codifica.

Il Capitolo 6 approfondisce la fase di verifica e validazione del progetto, specificando le modalità ed i risultati ottenuti.

Il Capitolo 7 riporta le conclusioni oggettive e soggettive a cui si è giunti per il progetto.

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

* gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
* per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*[g];
* i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati in *corsivo*.

Capitolo 2

Descrizione dello stage

Questo capitolo descrive in dettaglio lo stage. Ne specifica il progetto da svolgere contestualizzandolo nella realtà aziendale e, definendone i requisiti, gli obiettivi da raggiungere e la pianificazione iniziale.

2.1 Introduzione al progetto e contestualizzazione

Nel contesto del supply chain management in cui l'azienda opera, ha assunto una gran rilevanza l'organizzazione dei turni di singoli lavoratori o di squadre di lavoratori, in relazione alle competenze da loro possedute e richieste dai vari lavori da portare a compimento.

Il progetto di stage si prefiggeva l'obiettivo di estendere per la risoluzione dei problemi di scheduling il *framework*[g] aziendale e scrivere un'applicazione che lo utilizzasse, andando a risolvere un problema specifico. L'obiettivo finale è utilizzare il framework per lo scheduling delle squadre dei meccanici, tuttavia è stato deciso che, per cominciare a studiare i problemi di scheduling e la fattibilità di risolverli con metodi *euristici*[g] e *meta-euristici*[g], fosse opportuno cominciare da un problema più semplice: l'organizzazione dei turni ai tavoli di un casinò. L'organizzazione dei turni dei meccanici sarà il naturale proseguimento di questo progetto, e la complessità maggiore dovuta allo schedulazione di individui singoli inseriti in un gruppo anch'esso da schedulare sarà supportata dal framework di base, robusto e già testato, sviluppato all'interno del progetto di stage.
Il problema della schedulazione dei turni all'interno dei casinò è stato scelto come problema di partenza in quanto comunque di interesse per l'azienda, che ha contatti con un manager de "The Hippodrome Casino" di Londra, il quale ha sottoposto il problema che, al momento, viene risolto completamente a mano, dovendo tenere conto di numerosi vincoli e variabili.

2.2 Vincoli metodologici, temporali e tecnologici

È stato concordato con Trans-Cel Autotrasporti che lo stage fosse svolto presso la sede operativa della stessa, ad Albignasego, affinché io potessi essere inserita nel contesto

aziendale ed avere la possibilità di confrontarmi col resto del team di sviluppo in maniera immediata.
Il tutor aziendale ha richiesto di scrivere ogni giorno un breve report delle attività svolte, delle difficoltà incontrate, degli obiettivi raggiunti e delle considerazioni personali su un documento condiviso su *Google Drive*[g], ha inoltre richiesto che ogni mattina prima di cominciare a lavorare si svolgesse una breve riunione con tutto il team per focalizzarsi sulle attività della giornata.
Lo stage, che prevede una durata di 300 ore, è stato pianificato su una durata di poco meno di 9 settimane prevedendo in media 6.5 ore lavorative al giorno. Tale stima è stata effettuata al ribasso per permettere una certa flessibilità per quanto riguarda l'orario di lavoro e/o eventuali giorni liberi che avrei potuto prendere per impegni universitari; l'orario di lavoro invece è stato stabilito dal lunedì al venerdì, dalle 9.00 alle 18.00 con un'ora di pausa pranzo.
Prima dell'inizio dello stage Trans-Cel Autotrasporti ha redatto un piano di lavoro in cui sono stati fissati gli obiettivi e la pianificazione settimanale per lo stage, che vengono illustrati nei paragrafi "2.3 - Obiettivi" e "2.4 - Pianificazione" di questo capitolo.
Per quanto riguarda i vincoli tecnologici, essi saranno trattati nel paragrafo "2.5 - Ambiente di lavoro" di questo capitolo.

2.3 Obiettivi

Sono riportati nella Tabella 2.1 gli obiettivi fissati per lo stage con rispettivo identificativo, importanza e breve descrizione.

L'identificativo (in breve, Id) è la sigla che identifica ogni requisito e rispetta la notazione *[Importanza][Identificativo]*, dove l'importanza è la sigla **Ob** / **De** / **Op** a seconda che l'obiettivo sia obbligatorio, desiderabile o opzionale; l'identificativo è un numero progressivo che identifica in modo univoco l'obiettivo.

Un obiettivo è classificato, secondo l'importanza, come:

* **Obbligatorio:** è l'importanza attribuita agli obiettivi il cui soddisfacimento dovrà necessariamente avvenire in quanto sono di importanza fondamentale per la riuscita del progetto;
* **Desiderabile:** è l'importanza attribuita agli obiettivi il cui soddisfacimento non è necessario, tuttavia desiderabile;
* **Opzionale:** è l'importanza attribuita agli obiettivi il cui soddisfacimento è del tutto opzionale in quanto renderebbero il progetto più completo.

Nel paragrafo "7.1 - Raggiungimento degli obiettivi" è riportato un consuntivo che riporta quanti degli obiettivi riportati sono stati soddisfatti nel corso dello stage.

Tabella 2.1: Obiettivi da raggiungere a fine stage

ID	IMPORTANZA	DESCRIZIONE
Ob1	Obbligatorio	Assimilazione dei concetti di base dei problemi di scheduling
Ob2	Obbligatorio	Definizione del problema di scheduling da risolvere
Ob3	Obbligatorio	Studio della letteratura scientifica sui modelli e i metodi per problemi di schedulazione e indagine sugli algoritmi euristici più adatti al tipo di problema in esame
Ob4	Obbligatorio	Studio del framework aziendale dalla relativa documentazione
Ob5	Obbligatorio	Progettazione di un algoritmo euristico per la soluzione del problema
Ob6	Obbligatorio	Implementazione prototipale dell'algoritmo euristico
Ob7	Obbligatorio	Integrazione dell'algoritmo euristico nelle librerie dell'azienda
Ob8	Obbligatorio	Test preliminare su istanze benchmark di problemi di scheduling
Ob9	Obbligatorio	Produzione di documentazione sui moduli sviluppati
Ob10	Obbligatorio	Produzione di un manuale di utilizzo dei moduli sviluppati
De1	Desiderabile	Analisi prestazionale su diverse istanze del problema
De2	Desiderabile	Esplorazione di altri contesti applicativi
Op1	Opzionale	Definizione di un modello di programmazione matematica del problema
Op2	Opzionale	Implementazione con AMPL (o altro linguaggio di modellazione matematica) del modello di programmazione lineare intera per la soluzione esatta del problema di scheduling

2.4 Pianificazione

La pianificazione del lavoro da svolgere durante lo stage è stata costruita sulla base delle ore di lavoro previste dallo stage curricolare.
Il progetto è stato suddiviso in otto periodi:

1. **Analisi del problema.** Periodo di assimilazione dei concetti di base dei problemi di scheduling, per comprenderne contesto e applicazione; approccio al problema dello scheduling nei casinò, con definizione dei vincoli a cui si deve sottostare.

2. **Analisi dello stato dell'arte.** Periodo di approfondimento dello studio dei problemi di scheduling, sia nella teoria che nella pratica, con relativi metodi di risoluzione.

3. **Ideazione e progettazione di un algoritmo euristico.** Periodo di ricerca di algoritmi euristici per la risoluzione dei problemi di scheduling e di definizione, sulla base di questi ultimi, dello scheletro di un algoritmo per lo scheduling nei casinò.

4. **Implementazione dell'algoritmo euristico.** Periodo di familiarizzazione con il framework aziendale, di progettazione dell'architettura da integrare con lo stesso per estenderlo con la risoluzione dei problemi di scheduling, e conseguente implementazione.

5. **Integrazione e test dell'algoritmo.** Periodo di implementazione dell'applicazione per il problema specifico dello scheduling nei casinò utilizzando il framework aziendale già esteso; si sono inoltre svolti i primi test semplificati.

6. **Confronto con tecniche esatte.** Periodo di studio di modelli matematici per problemi di scheduling classici e definizione di un modello matematico per lo scheduling nei casinò, con conseguente implementazione in *AMPL*[g] e confronto fra *metodo esatto*[g] e euristica prodotta.

7. **Verifica e testing.** Periodo trasversale ai periodi di Implementazione dell'algoritmo euristico, Integrazione e test dell'algoritmo, Confronto con tecniche esatte. Prevede il testing del prodotto.

8. **Produzione di documenti, manuali e relazioni.** Periodo trasversale ai periodi di Ideazione e progettazione di un algoritmo euristico, Implementazione dell'algoritmo euristico, Integrazione e test dell'algoritmo, Confronto con tecniche esatte, Verifica e testing. Prevede la scrittura di documentazione per il codice, comprendente il manuale sviluppatore e il manuale utente; la scrittura di relazioni sui test effettuati e sullo studio del problema.

Nella Tabella 2.2 vengono riportate in dettaglio le attività previste per ogni periodo, specificando il numero di ore preventivate per ognuna e il periodo in cui ne è pianificato lo svolgimento. Nella Figura 2.1 e nella Figura 2.2 è riportato inoltre il *diagramma di Gantt*[g] della stessa pianificazione.

Nel paragrafo "7.3 - Consuntivo finale" è riportato un consuntivo che confronta le ore preventivate con le ore effettivamente dedicate ad ogni attività.

Tabella 2.2: Pianificazione del periodo di stage

ATTIVITÀ	ORE	DAL	AL
1. Analisi del problema			
Assimilazione dei concetti di base per problemi di scheduling	10	10/07	11/07
Definizione delle caratteristiche del problema di scheduling da risolvere	10	11/07	12/07
2. Analisi dello stato dell'arte			
Studio della letteratura di base su problemi di scheduling	15	13/07	14/07
Ricerca ed analisi degli algoritmi più adatti	15	17/07	18/07
3. Ideazione e progettazione di un algoritmo euristico			
Individuazione dei blocchi di base da algoritmi pre-esistenti	15	19/07	20/07
Integrazione dei blocchi	10	21/07	24/07
Definizione dell'algoritmo di soluzione	15	24/07	25/07
4. Implementazione dell'algoritmo euristico			
Studio ed assimilazione del framework aziendale	15	26/07	27/07
Progettazione dei moduli	15	28/07	31/07
Implementazione	40	01/08	07/08
5. Integrazione e test dell'algoritmo			
Integrazione dell'algoritmo nelle librerie dell'azienda	30	08/08	11/08
Test preliminari su istanze semplificate	15	21/08	22/08
6. Confronto con tecniche esatte			
Studio dei modelli in letteratura	10	23/08	24/08
Definizione di un modello matematico	10	24/08	25/08
Implementazione del modello in AMPL	10	28/08	29/08
Confronto con i risultati dell'euristica e analisi delle presentazioni	10	29/08	30/08
7. Verifica e testing	20	01/08	04/09
8. Produzione di documenti, manuali e relazioni	35	17/07	15/09

Figura 2.1: Diagramma di Gantt per il piano di lavoro, settimane 1-5

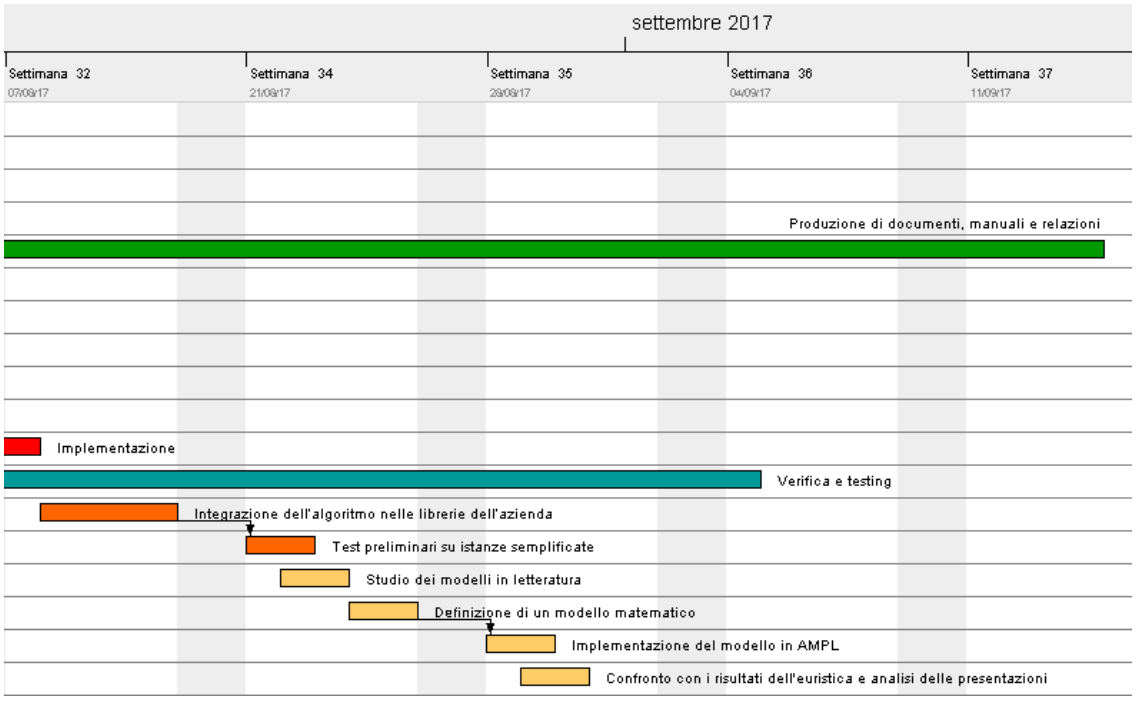


Figura 2.2: Diagramma di Gantt per il piano di lavoro, settimane 5-9

2.5 Ambiente di lavoro

2.5.1 Metodologia di sviluppo

La metodologia di sviluppo adottata in Trans-Cel Autotrasporti non si riconduce ad un modello in particolare, nonostante presenti delle affinità sia con il *modello incrementale*[g] che con il metodo *agile*[g] di tipo *scrum*[g]. Infatti, per la pianificazione del lavoro a lungo termine sono stati definite delle fasi, affini idealmente a degli incrementi, associate a *milestones*[g] ben definite; tuttavia all'interno di ogni incremento l'approccio allo sviluppo diventa più simile al metodo agile, in quanto il Project Manager, ma anche i membri del team di sviluppo stesso, inseriscono e assegnano/si assegnano i task del backlog. Affine al metodo scrum è anche il controllo quotidiano dell'avanzamento dei lavori rappresentato dalla breve riunione mattutina con tutti i membri del team.
Questa metodologia di sviluppo funziona all'interno di Trans-Cel Autotrasporti poiché il team sviluppa un software per l'azienda stessa e quindi il colloquio e l'interazione fra sviluppatori e stakeholders è estremamente facilitato.

2.5.2 Gestione di progetto

Versionamento

Per quanto riguarda il versionamento, l'azienda ha un *repository*[g] *Git*[g] privato su *Bitbucket* [3], a cui mi è stato dato l'accesso in lettura. Il ramo master di tale repository costituisce il codice del framework e delle applicazioni aggiornato all'ultima versione e funzionante; mentre per tutte le modifiche in via di sviluppo o le estensioni i membri del team lavorano su dei branch personali. Anche per me ne è stato creato uno, all'interno del quale ho potuto modificare il framework e scrivere la mia applicazione. Nonostante non avessi mai lavorato su Bitbucket, non ho trovato problemi nell'adattarmi in quanto molto simile ad altri *repository manager* quali *GitLab* [10] o *GitHub* [9], rispetto ai quali presenta tra l'altro diversi vantaggi: supporta infatti sia repository Git che *Mercurial*[g] e permette di avere repository private illimitate per team al di sotto dei cinque componenti.
Come *Graphical User Interface (GUI)*[g] per Git (interfaccia grafica per la gestione dei repository) è stato utilizzato SourceTree, in quanto è il client raccomandato per BitBucket. Offre una visualizzazione grafica dei branch che ne permette una comprensione immediata e, nonostante la semplicità dell'interfaccia, espone tutti i comandi base di Git in maniera intuitiva.

Ticketing

Le attività da svolgere nel tempo medio-breve, che vanno a costituire il backlog, sono gestite tramite la piattaforma di project management *Taiga* [16]. I membri del team possono aggiungere attività al backlog che inizialmente ricadono nella categoria "to do", quando vengono assegnate a qualcuno che incomincia a occuparsene nella categoria "In progress" e poi successivamente in "To test" e "Complete". Alla stessa attività possono venire assegnati più membri del team. Anche per me è stata creata un'utenza con accesso ai ticket.

Altri strumenti per la gestione di progetto

* *Google Drive* [12] per la condivisione veloce di file come guide, documentazioni, riferimenti utili e documenti informali.

* *Google Calendar* [11] per la gestione degli eventi ed impegni.

2.5.3 Documentazione

Per quanto riguarda la produzione di documenti, manuali e relazioni richiesti, l'azienda non ha uno standard e mi ha dunque delegato la scelta di che software utilizzare per la loro produzione.

Doxygen

Per quanto riguarda la documentazione del codice, la mia scelta è ricaduta su *Doxygen* [6]. Doxygen è uno strumento per la generazione automatica di documentazione a partire dal codice commentato per molti linguaggi di programmazione, fra i quali il C++. Genera documentazione sia in HTML che in LaTeX, corredandola con diagrammi delle classi ed estraendo i commenti direttamente dai sorgenti per documentare attributi e metodi delle classi.
Doxygen è uno strumento semplice da usare e installare e produce una documentazione elegante e ben strutturata. Un vantaggio che ho trovato nell'utilizzarlo (producendo la documentazione in HTML) è che, avendo tutti i file html, css e javascript necessari disponibili sia lettura che scrittura è semplice da manipolare per fare dei piccoli cambiamenti utili alla documentazione prodotta.
Un'altro strumento preso in considerazione per la produzione della documentazione è stato Doc++; tuttavia è stato scartato in quanto la struttura dei commenti da scrivere per la produzione della documentazione è più complicata di quella da adottare con Doxygen e rende i commenti all'interno dei sorgenti di difficile lettura. Produce inoltre una documentazione meno curata di quanto non faccia Doxygen.

Diagrammi UML e Use Case

Per la progettazione è stato necessario modellare con UML, la mia scelta è ricaduta su *Astah Professional* [2], disponibile in versione gratuita grazie alla licenza accademica, perché identificato come il software più completo e di facile utilizzo fra quelli dedicati (ad esempio, GNU Dia o StarUML). Astah Professional supporta sia i costrutti di UML 1.x che di UML 2.x e permette di modellare casi d'uso, i diagrammi delle classi, degli oggetti, delle attività e di sequenza; produce inoltre diagrammi di aspetto semplice ed essenziale adatti ad essere inseriti in relazioni formali.

2.5.4 Ambiente di sviluppo

Il framework aziendale da estendere è programmato in C++ ed offre una base per l'implementazione di algoritmi di ottimizzazione. Consiste di librerie per l'utilizzo di grafi, di algoritmi euristici di tipo *greedy*[g] e di meta-euristiche, ad esempio *Tabu Search*[g], che l'estensione del framework andrà a sfruttare.

IDE

Il progetto originale è nato su Visual Studio, perciò è consigliato utilizzare tale *IDE*[g] per lo sviluppo in quanto i file di progetto con estensione *.vcxproj* che contengono le informazioni necessarie alla compilazione sono già presenti.
Trattandosi comunque di codice scritto in C++, può essere utilizzato qualsiasi IDE ne fornisca il supporto.

Sistemi operativi

Poiché il progetto è nato su Visual Studio, la scelta più naturale per il sistema operativo è rappresentata da Windows; ciononostante i membri del team possono lavorare indifferentemente anche su Linux o Mac OS.

Poiché durante lo stage ho lavorato su un computer con installato solamente Windows come sistema operativo, il problema della scelta non si è posto e ho utilizzato Visual Studio come IDE.

2.6 Analisi preventiva dei rischi

Nella Tabella 2.3 viene riportata l'analisi dei rischi in cui si può incorrere nel corso del progetto.
Nel paragrafo "7.2 - Resoconto dell'analisi dei rischi" è riportata un'attualizzazione dei rischi che si sono effettivamente verificati.

Tabella 2.3: Analisi preventiva dei rischi

Descrizione	Trattamento	Rischio
Scelte di bad design nella progettazione del framework aziendale esistente, che possono rendere difficoltosa l'estensione.	Se verificato, si potrà avere un confronto con gli sviluppatori del framework per cercare assieme una soluzione.	Occorrenza: Alta
		Pericolosità: Medio-Alta
Bug all'interno del framework poiché alcune sue parti si trovano ancora in fase di sviluppo.	Se verificato, si potrà avere un confronto con gli sviluppatori del framework per cercare assieme una soluzione.	Occorrenza: Bassa
		Pericolosità: Alta
Difficoltà nel confrontarsi con algoritmi euristici e problemi di ricerca operativa di cui si ha conoscenza nulla/scarsa	Se verificato, si dedicheranno più ore del previsto allo studio per rafforzare le basi.	Occorrenza: Bassa
		Pericolosità: Bassa

Capitolo 3

Definizione del problema

Questo capitolo espone i risultati ottenuti durante i primi tre periodi previsti dal piano di lavoro; al suo interno viene quindi approfondito l'argomento dei problemi di scheduling e viene definito nei dettagli il problema da risolvere durante lo stage. Le informazioni qui presentate sono frutto di uno studio della letteratura di base per i problemi di scheduling: per la sua stesura sono stati consultati i paper [1, 4, 7, 13, 17].

3.1 Problemi di scheduling

Con il termine *problemi di scheduling* si intende una vasta classe di problemi decisionali, molto diversi tra loro per complessità e struttura, in cui riveste importanza il fattore tempo visto come risorsa (scarsa) da allocare in modo ottimo a determinate attività; essi quindi ruotano attorno alle modalità di assegnamento all'interno di slot temporali (*TTB*, "Time Table blocks") di una risorsa (*Task*) ad un'attività che deve essere effettuata (*Item*).
Una soluzione di un problema di scheduling prende il nome di *schedule*. In termini generali, una schedule è una descrizione completa dell'utilizzo temporale dei *Task* che devono essere eseguiti dagli *Item*; nella risoluzione di un problema di schedulazione siamo interessati a schedule ammissibili, cioè che rispettino tutti i vincoli impliciti in qualsiasi problema di scheduling, quali ad esempio che un *Task* non può essere occupato da due *Item* contemporaneamente o che un *Item* non possa essere assegnato a due *Task*, ma anche altri vincoli determinati dal problema specifico.
I vincoli all'interno di un problema di scheduling vengono classificati all'interno di due categorie:

* **Hard constraints:** sono i vincoli inviolabili per il problema, se uno di essi non viene rispettato all'interno di una soluzione, tale soluzione non è ammissibile. Sono vincoli dati dalla natura degli *Item* da schedulare o imposti da fattori esterni, come leggi in merito o regolamenti.

* **Soft constraints:** sono i vincoli che possono essere violati all'interno di una soluzione, producendo ciononostante una soluzione ammissibile. Sono vincoli legati al concetto di *fairness*: produrre una soluzione "fair" consiste nel bilanciare il più possibile fra gli *Item* i *Task* o i *TTB* ritenuti "impopolari".

Degni di nota all'interno dell'insieme degli *hard constraints* sono i *sequence constraints*, che per ogni *TTB* b definiscono un insieme $\mathcal{T}_\mathrm{b}$ di *TTB* che possono seguire b per un certo *Item*.

Fra tutte le soluzioni ammissibili, chiaramente alcune sono migliori di altre. Il concetto di "migliore" viene definito grazie ai *costi* associati ad ogni assegnamento di un *Item* ad un *Task* per un certo *TTB*, e a delle *penalità* associate ai *soft constraints*. Ciò che mette insieme costi e penalità è la *funzione obiettivo* il cui valore è un numero, chiamato obiettivo, confrontabile con quello delle altre soluzioni ammissibili per determinare la migliore, ovvero quella che utilizza gli *Item* il più efficientemente possibile.

3.2 Definizione formale degli elementi generali per i problemi di scheduling

3.2.1 Dati

Item s denota un *Item*, l'insieme di tutti gli *Item* è indicato con $\mathcal{S}$

TTB b denota un *TTB*, l'insieme di tutti i *TTB* è indicato con $\mathcal{B}$

Task t denota un *Task*, l'insieme di tutti i *Task* è indicato con $\mathcal{T}$

Costi c_{sbt} denota il costo associato all'assegnamento di un *Item* s ad un *Task* t durante il *TTB* b

Variabili decisionali $\mathcal{X}$ denota l'insieme delle variabili decisionali x_{sbt} t.c.

$$x_{\mathrm{sbt}} = \begin{cases} 1, & \text{se l'}\textit{Item}\ s \text{ è assegnato al } \textit{Task}\ t \text{ per il } \textit{TTB}\ b \\ 0, & \text{altrimenti} \end{cases}$$

3.2.2 Vincoli

Hard ogni *Item* s non può essere assegnato a più di un *Task* t, *TTB* b contemporaneamente.

$$\sum_{b \in \mathcal{B},\ t \in \mathcal{T}} x_{\mathrm{sbt}} \leq 1, \forall\ s \in \mathcal{S}$$

Soft c denota un soft constraint, f_c è la funzione che assegna una penalità a c, $f_c(X)$ è la penalità della soluzione X. L'insieme di tutti i soft constraints è indicato con $\mathcal{C}$.

Sequence $\mathcal{T}_\mathrm{b}$ denota l'insieme di *TTB* che possono essere schedulate dopo la *TTB* b.

$$\sum_{b' \in \mathcal{T}_b} x_{\mathrm{sb't}} \geq x_{\mathrm{sbt}}, \forall\ s \in \mathcal{S},\ b \in \mathcal{B},\ t \in \mathcal{T}$$

3.2.3 Obiettivo

Costi il costo C di una soluzione X è la somma dei costi per ogni assegnamento:

$$C = \sum_{s \in \mathcal{S},\, b \in \mathcal{B},\, t \in \mathcal{T}} x_{\mathrm{sbt}}\, c_{\mathrm{sbt}}$$

Fairness data la misura di popolarità u_b e u_t per ogni *TTB* b, per ogni *Task* t; dato α_s coefficiente per tenere in conto il carico di lavoro del singolo *Item*, l'impopolarità della schedule per un *Item* s può essere ora definita come:

$$U_{\mathrm{s}} = \alpha_{\mathrm{s}} \sum_{b \in \mathcal{B},\, t \in \mathcal{T}} (x_{\mathrm{sbt}}\, u_{\mathrm{b}} + x_{\mathrm{sbt}}\, u_{\mathrm{t}})$$

possiamo stabilire una misura generale di fairness

$$C_{\mathrm{fair}} = \max U_s - \min U_s$$

Penalità stabiliamo il costo delle violazioni dei *soft constraints* come:

$$C_{\mathrm{soft}} = \sum_{c \in \mathcal{C}} f_{\mathrm{c}}(X)$$

Funzione obiettivo per ogni soluzione X:

$$F(X) = C(X) + C_{\mathrm{fair}}(X) + C_{\mathrm{soft}}(X)$$

3.3 Definizione del problema: scheduling nei casinò

3.3.1 Analisi discorsiva

All'interno del casinò bisogna organizzare i turni per tutte le postazioni lavorative aperte, coprendo l'orizzonte temporale che parte da mezzogiorno e finisce alle sei del mattino dopo.
Le postazioni sono di due tipi: i tavoli, ai quali si siedono i giocatori; e i pit, che sono delle postazioni di controllo sui tavoli da cui lavoratori più qualificati sovrintendono il comportamento di lavoratori e giocatori nei tavoli a loro assegnati.
Esistono tre categorie di lavoratori: i *dealer*, che possono lavorare nei tavoli; gli *inspector*, che possono lavorare nei pit; i *dealer-inspector* che possono lavorare in entrambe le tipologie di postazione. I dealer hanno un livello da 8 a 1 a seconda della loro esperienza: ogni tavolo richiede un certo livello di esperienza per potervi lavorare. Gli inspector hanno a loro volta un livello da 3 a 1: come succede per i tavoli, anche ogni pit può richiedere un certo livello di inspector. Inoltre ogni postazione richiede la conoscenza di un determinato gioco; i dealer e gli inspector differiscono per conoscenza di giochi, quindi ci sono postazioni in cui possono o non possono lavorare.
Lo scheduler deve fornire una schedule in tempi brevi per assicurare un'utilità in decisioni volubili durante l'arco della giornata, poiché la richiesta di copertura dei tavoli dipende dai clienti che arrivano.
Il turno di un lavoratore è diviso in scaglioni da 20 minuti, il numero totale di ore lavorate in una giornata dipende da ciascun lavoratore; inoltre ci possono essere degli straordinari che vengono decisi dal manager durante la giornata (i lavoratori possono

accettarli o meno) fino a 8 ore.
Dopo cinque scaglioni consecutivamente lavorati, un lavoratore deve obbligatoriamente avere un turno di pausa, tuttavia è preferibile tenere la media di turni consecutivi intorno ai tre/quattro. Inoltre, dopo ogni pausa il lavoratore non può tornare a lavorare nella stessa postazione dove prestava servizio precedentemente.
L'obiettivo è coprire tutte le postazioni aperte con un lavoratore disponibile e bilanciare per quanto possibile lo scheduling in materia di turni consecutivi e postazioni in cui si lavora seduti o in piedi.

3.4 Modello per lo scheduling nei casinò

3.4.1 Dati

Item sono i lavoratori *s*.
Per la distizione fra dealer, inspector e dealer-inspector si adottano i parametri I_s e D_s t.c.

$$I_{\mathrm{s}} = \begin{cases} 1, & \text{se il lavoratore } s \text{ è un inspector} \\ 0, & \text{se il lavoratore } s \text{ è un dealer/dealer-inspector} \end{cases}$$

$$D_{\mathrm{s}} = \begin{cases} 1, & \text{se il lavoratore } s \text{ è un dealer} \\ 0, & \text{se il lavoratore } s \text{ è un inspector/dealer-inspector} \end{cases}$$

Inoltre si definiscono i seguenti parametri:

$$presente_{\mathrm{sb}} = \begin{cases} 1, & \text{se il lavoratore } s \text{ è può lavorare durante lo scaglione b} \\ 0, & \text{altrimenti} \end{cases}$$

$$straordinari_{\mathrm{sb}} = \begin{cases} 1, & \text{se il lavoratore } s \text{ sta svolgendo straordinari durante lo scaglione b} \\ 0, & \text{altrimenti} \end{cases}$$

$$livello\_l_{\mathrm{s}} = \begin{cases} 10-3, & \text{se il lavoratore è un dealer} \\ 3, & \text{se il lavoratore è un dealer-inspector} \\ 3-1, & \text{se il lavoratore è un inspector} \end{cases}$$

$$giochi\_l_{\mathrm{sg}} = \begin{cases} 1, & \text{se il gioco g è conosciuto dal lavoratore s} \\ 0, & \text{altrimenti} \end{cases}$$

TTB sono gli scaglioni temporali *b* da 1 a 54. Si definisce il seguente parametro:

$$popularity_{\mathrm{t}} = \begin{cases} 1, & \text{se lo scaglione b non è popolare} \\ 0, & \text{altrimenti} \end{cases}$$

Task sono le postazioni da occupare *t*. Si definiscono i seguenti parametri:

$$livello\_p_{\mathrm{t}} = \begin{cases} 10-3, & \text{se la postazione t è un tavolo} \\ 3-1, & \text{altrimenti} \end{cases}$$

$$giochi\_p_{\mathrm{tg}} = \begin{cases} 1, & \text{se il gioco g è richiesto alla postazione t} \\ 0, & \text{altrimenti} \end{cases}$$

$$popularity_{\mathrm{t}} = \begin{cases} 1, & \text{la postazione t richiede di stare in piedi} \\ 0, & \text{altrimenti} \end{cases}$$

$$aperto_{\mathrm{t}} = \begin{cases} 1, & \text{la postazione t è aperta} \\ 0, & \text{altrimenti} \end{cases}$$

$$k = \text{numero di tavoli}$$

Costi c_{st} denota il costo associato all'assegnamento di un lavoratore s durante uno scaglione b (gli straordinari hanno una maggiorazione dello stipendio)

Giochi viene inoltre definito l'insieme $\mathcal{G}$ di giochi g.

Variabili decisionali $x, z \in \mathcal{X}$ t.c.

$$x_{\mathrm{sbt}} = \begin{cases} 1, & \text{se il lavoratore } s \text{ è assegnato alla postazione } t \text{ per lo scaglione } b \\ 0, & \text{altrimenti} \end{cases}$$

$$z_{\mathrm{s}} = \text{numero di turni lavorati consecutivamente dal lavoratore s}$$

3.4.2 Vincoli

Hard

* Lavoratore s può fare turno solo se presente in sala e non può fare più di un turno alla volta:
$$\sum_{t \in \mathcal{T}} x_{\mathrm{sbt}} \leq presente_{\mathrm{sb}}, \forall s \in \mathcal{S},\ b \in \mathcal{B}$$

* Tutte le postazioni t aperte devono avere qualcuno che ci lavori:
$$\sum_{s \in \mathcal{S}} x_{\mathrm{sbt}} = aperto_{\mathrm{t}}, \forall b \in \mathcal{B},\ t \in \mathcal{T}$$

* Un inspector non può lavorare in un tavolo:
$$\sum_{t \leq k,\, t \in \mathcal{T}} I_{\mathrm{s}}\, x_{\mathrm{sbt}} = 0, \forall s \in \mathcal{S},\ b \in \mathcal{B}$$

* Un dealer non può lavorare in un pit:
$$\sum_{t > k,\, t \in \mathcal{T}} D_{\mathrm{s}}\, x_{\mathrm{sbt}} = 0, \forall s \in \mathcal{S},\ b \in \mathcal{B}$$

* Ogni postazione t richiede un lavoratore con un livello uguale o più alto al proprio:
$$x_{\mathrm{sbt}}\, livello\_l_{\mathrm{s}} \leq livello\_p_{\mathrm{t}}, \forall s \in \mathcal{S},\ b \in \mathcal{B},\ t \in \mathcal{T}$$

* Ogni postazione t richiede un lavoratore che conosca il gioco:
$$x_{\mathrm{sbt}}\, giochi\_p_{\mathrm{tg}} = giochi\_l_{\mathrm{st}}, \forall s \in \mathcal{S},\ b \in \mathcal{B},\ t \in \mathcal{T},\ g \in \mathcal{G}$$

Soft

* I turni dei lavoratori preferibilmente devono essere di tre/quattro scaglioni consecutivi:

$$z_{\mathrm{s}} \geq \sum_{b \in \mathcal{B},\, t \in \mathcal{T}} x_{\mathrm{sbt}}, \forall s \in \mathcal{S}$$

* Fairness per il numero di scaglioni in piedi/seduti:

$$fair_{\mathrm{s}} = \sum_{b \in \mathcal{B},\, t \in \mathcal{T}} x_{\mathrm{sbt}}\, popularity_{\mathrm{t}}, \forall s \in \mathcal{S}$$

Sequence

* Dopo cinque scaglioni lavorati consecutivamente, uno deve essere di pausa:

$$\sum_{b=m \ldots m+5} \sum_{t \in \mathcal{T}} x_{\mathrm{sbt}} \leq 5, \forall s \in \mathcal{S},\ m = 1 \ldots 49$$

* Dopo cinque scaglioni lavorati e una pausa, il lavoratore deve cambiare tavolo:

$$\sum_{b=m \ldots m+6} \sum_{t \in \mathcal{T}} x_{\mathrm{sbt}} \leq 5, \forall s \in \mathcal{S}, m = 1 \ldots 48$$

* Non possono esserci due pause consecutive:

$$\sum_{b=m \ldots m+1} \sum_{t \in \mathcal{T}} x_{\mathrm{sbt}} \geq 1, \forall s \in \mathcal{S}, m = 1 \ldots 53$$

3.4.3 Obiettivo

$$\min \left(\sum_{s \in \mathcal{S}} \left(\sum_{b \in \mathcal{B},\, t \in \mathcal{T}} (straordinari_{\mathrm{sb}}\, c_{\mathrm{sbt}}) + z_{\mathrm{s}} + fair_{\mathrm{s}} \right) \right)$$

3.5 Algoritmi euristici di risoluzione per problemi di scheduling

Nella letteratura è difficile trovare un algoritmo che vada a creare uno scheduling iniziale per i problemi di schedulazione in maniera veloce, i paper si concentrano spesso su metaeuristiche che vadano infatti a migliorare una possibile soluzione iniziale già presente.

Nel paper [4] sono nominate tre diverse metodologie per trovare uno scheduling iniziale: la prima genera tutte le soluzioni in maniera random andando poi a tenere in considerazione solo le soluzioni ammissibili, la seconda genera le soluzioni euristicamente (ma non viene specificato come), la terza strategia genera invece una parte della soluzione utilizzando un'euristica e la restante parte in maniera randomica. Una volta ottenuta in questo modo una schedule iniziale, viene applicata una *Scatter Search*[g] (metaeuristica).

Basandosi invece sul paper [7], per creare la schedule iniziale viene utilizzato un

algoritmo Greedy. L'obiettivo è ordinare, tramite una funzione di ordinamento euristica, tutti i turni secondo la difficoltà stimata di assegnazione o quanto è probabile che tale turno causi alta penalità. Una volta che i turni sono stati ordinati, sono assegnati in ordine: in questa maniera, i turni più complicati sono assegnati per primi nel processo di costruzione della schedule. Uno alla volta, i turni sono assegnati a tutte le possibili infermiere e si calcola la penalità in cui si incorre compiendo tale assegnazione; il turno è assegnato quindi nella maniera che garantisca la penalità più bassa (le penalità consistono in turni unpopular, tipo turno di notte, al weekend, numero di persone capaci di fare quel lavoro e consistono in un *weight* assegnato ad ogni turno). Si passa poi a valutare il turno successivo.
Una volta ottenuta in questo modo una schedule iniziale, viene applicata una *Variable Neighbourhood Search*[g] (metaeuristica).

Tenendo conto di queste informazioni, al fine di estendere il framework aziendale con la risoluzione dei problemi di scheduling (con applicazione al problema del casinò) e creare una schedule iniziale ammissibile, ho considerato di:

* Creare una soluzione random;
* Utilizzare un algoritmo greedy;
* Utilizzare in un primo momento un algoritmo greedy, per poi completare la soluzione in maniera randomica.

L'ipotesi della soluzione random è stata ad un primo impatto scartata in quanto i vincoli di cui tenere conto sono numerosi e molto stringenti, cercare una soluzione random sembra perciò dispersivo e poco efficiente; si è optato quindi per utilizzare un algoritmo greedy ispirato al Nurse Rostering.
L'opzione di completare la soluzione in maniera randomica una volta assegnati i turni più complicati è stata tenuta in considerazione, ma non è stata implementata durante il corso dello stage per mancanza di tempo.

Nel dettaglio, l'algoritmo di risoluzione implementato per andare a creare la migliore schedule possibile è stato il seguente (presentato come diagramma di attività nella Figura 3.1):

1. Per ogni *TTB*, per ogni *Task*, valutare quanti sono gli *Item* disponibili ad essere schedulati. Ordinare quindi per priorità i *Task* da assegnare all'interno del *TTB*.
2. Se esiste un *Task* con nessun *Item* disponibile per la schedulazione, è impossibile trovare una soluzione ammissibile. L'algoritmo si interrompe.
3. Vengono valutati tutti i possibili *Item* per il primo *Task* in ordine di priorità. L'*Item* che garantisce il minor obiettivo viene schedulato per il *Task* durante il *TTB*. È questa la "scelta greedy" dell'algoritmo.
4. Se non ancora tutti i *Task* sono stati schedulati per il *TTB*, viene ricalcolata la priorità di assegnamento dei *Task* alla luce dell'assenza di disponibilità dell'ultimo *Item* schedulato. Si ripete dal punto 2.
5. Quando tutti i *Task* del *TTB* sono stati assegnati ad un *Item*, si procede con l'assegnare i *Task* del prossimo *TTB*, se esiste. Si ripete dal punto 1.
6. Se sono stati assegnati tutti i *Task* dell'ultimo *TTB*, si è trovata una soluzione ammissibile.

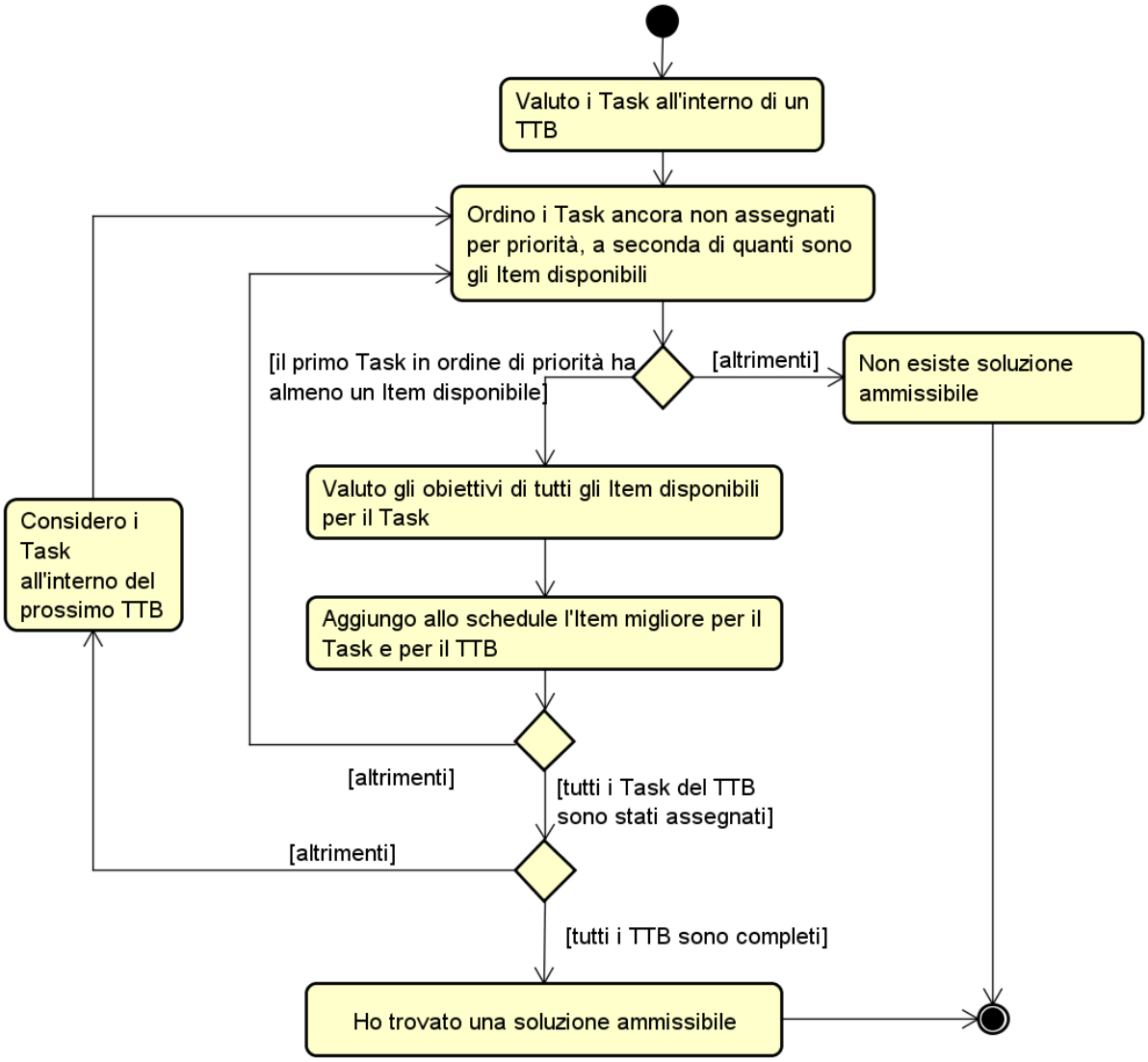


Figura 3.1: Diagramma di attività per l'algoritmo greedy di risoluzione per i problemi di scheduling

Si presenta tuttavia un problema: nel caso in cui gli *Item* non siano abbastanza da coprire tutti i *Task* durante i *TTB*, questo algoritmo si blocca e non produce alcuno scheduling. Sapendo che tramite l'euristica si vuole creare uno scheduling iniziale da ottimizzare poi tramite metodi metaeuristici, il non riuscire a produrre uno scheduling iniziale potrebbe limitare l'efficacia delle stesse metaeuristiche.

Nel paper [17], per arrivare sempre a una soluzione ammissibile viene introdotto un *Dummy Worker*[g], che viene poi eliminato tramite la Neighborhood Search (metaeuristica).

L'algoritmo di risoluzione implementato viene quindi lievemente modificato, sostituendo

al secondo punto "Se esiste un *Task* con nessun *Item* disponibile per la schedulazione, il *Task* viene assegnato al Dummy Worker". Il risultante algoritmo è schematizzato in Figura 3.2.

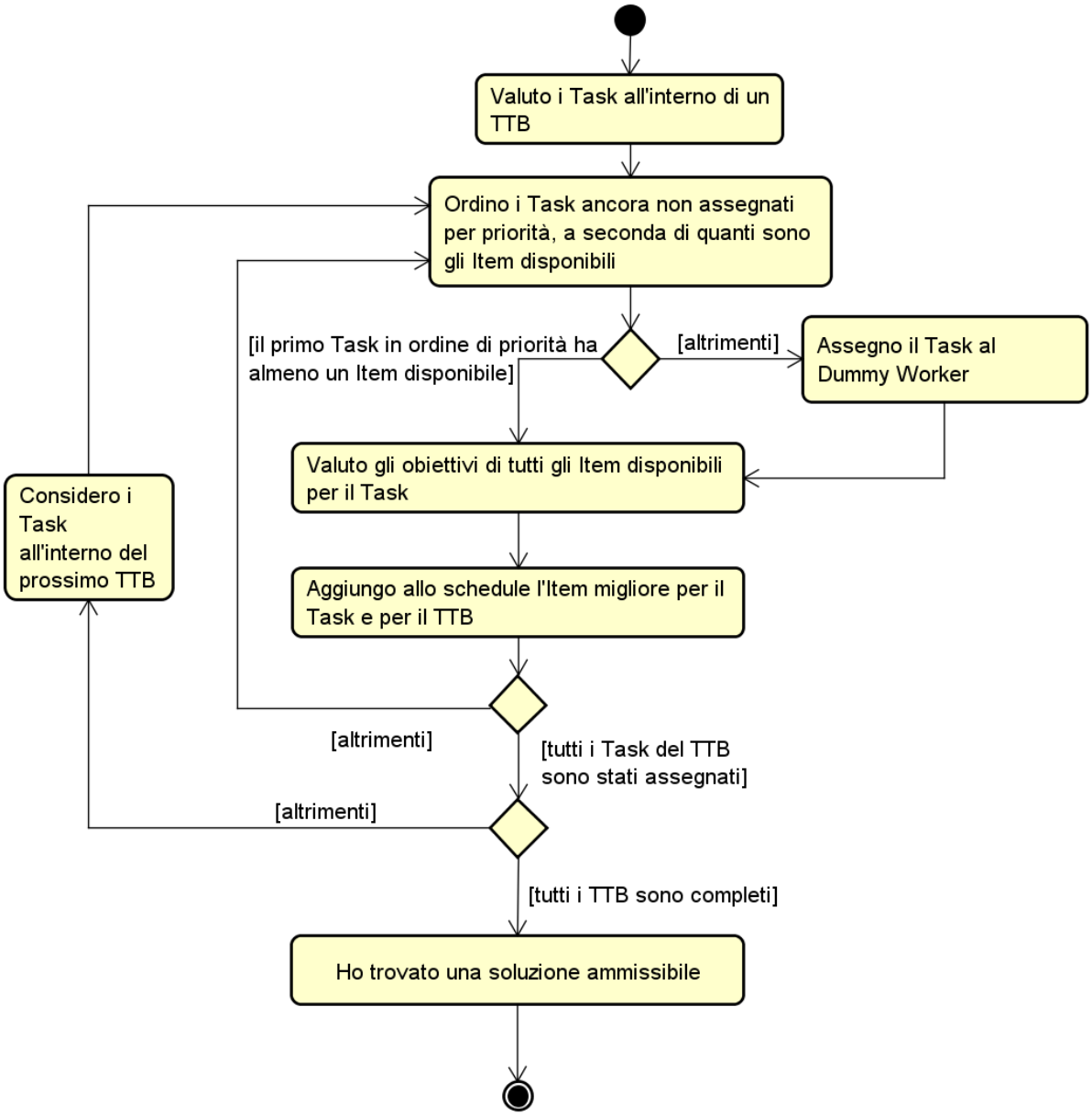


Figura 3.2: Diagramma di attività per l'algoritmo greedy di risoluzione per i problemi di scheduling con aggiunta di Dummy Worker

Capitolo 4

Analisi dei requisiti

In questo capitolo vengono definite con precisione le funzionalità del software che è stato prodotto durante lo stage. Viene presentata l'analisi dei requisiti svolta per il progetto, approfondita con i diagrammi dei casi d'uso.

4.1 Casi d'uso

I diagrammi dei casi d'uso (in inglese, *Use Case Diagram*) sono diagrammi di tipo UML che servono a descrivere le interazioni fra il sistema software e gli utenti che lo utilizzano, mostrando l'insieme funzionalità esposte dal sistema dal punto di vista degli utenti.
Poiché lo stage era incentrato sull'estensione del framework e lo sviluppo di un algoritmo euristico per la risoluzione dei problemi di scheduling, la prima parte del progetto non necessitava di esporre funzionalità all'utente. D'altro canto, una volta sviluppata l'applicazione per risolvere il problema specifico del casinò, avrei dovuto garantire dei servizi minimi all'utente per inserire l'input (informazioni circa le postazioni, i lavoratori, i turni) e mostrare l'output (lo scheduling prodotto).
Per questo motivo i diagrammi dei casi d'uso risultano minimali e in numero ridotto.

Ogni caso d'uso è classificato secondo la seguente convenzione:

UC[Codice padre]*.[Codice identificativo]

- **Codice padre**: il codice identificativo del caso d'uso generico che ha generato il caso d'uso in esame. Se il caso d'uso non è stato generato da altri, va tralasciato;
- **Codice identificativo**: Identifica univocamente il caso d'uso. È un codice composto da sole cifre.

Alcuni casi d'uso possono essere associati ad un diagramma dei casi d'uso riportante lo stesso titolo e codice.

4.1.1 Diagrammi dei casi d'uso

La Figura 4.1 presenta il diagramma dei casi d'uso generale.

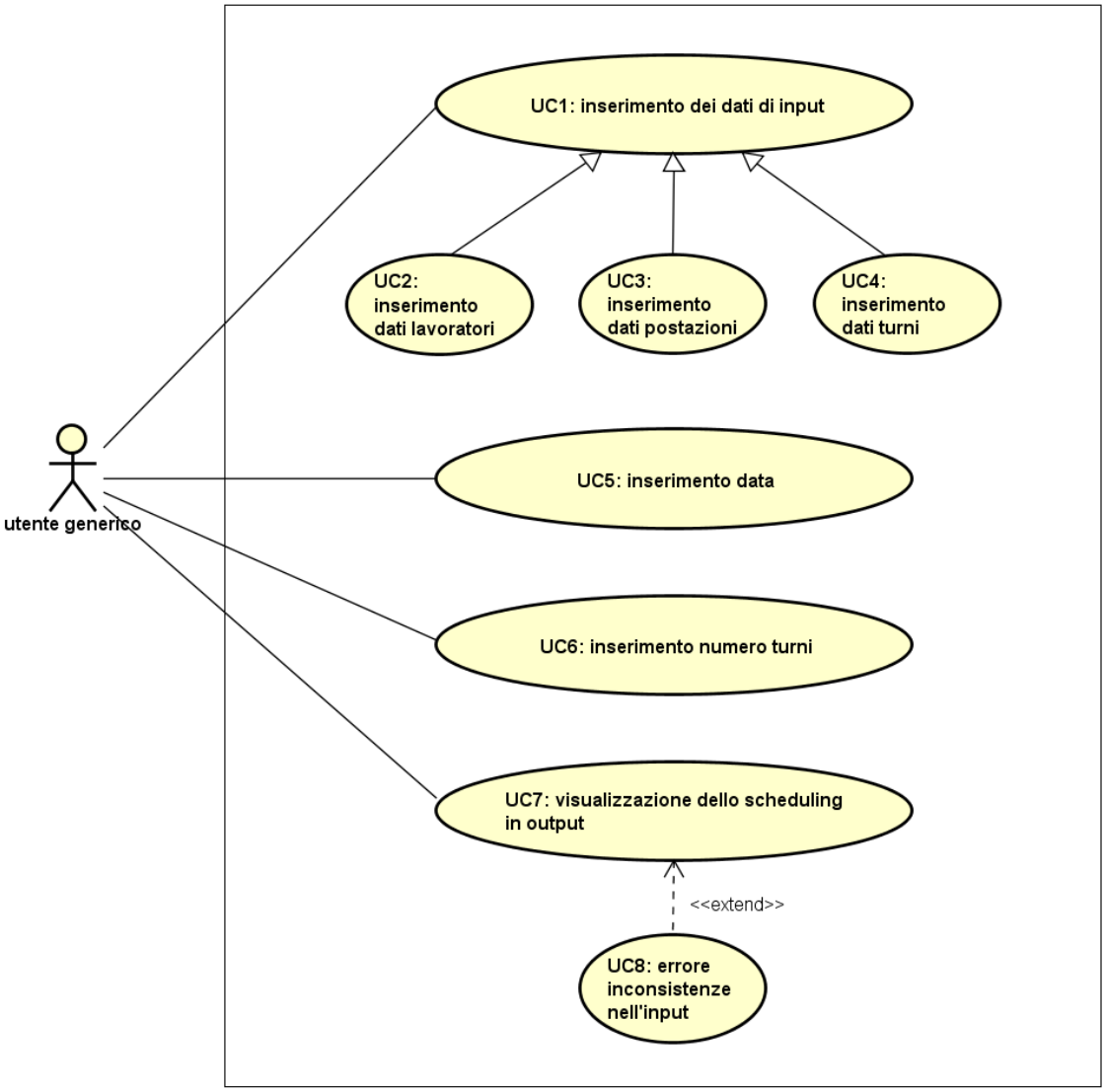


Figura 4.1: Diagramma dei casi d'uso ad alto livello

4.1.2 UC1: Inserimento dei dati di input

* **Attori**: utente generico;
* **Descrizione**: l'utente inserisce i dati in input al programma per realizzare uno scheduling.
* **Generalizzazioni**: l'utente inserisce i dati in input al programma per realizzare uno scheduling riguardati i lavoratori (UC2), le postazioni (UC3), i turni (UC4).

4.1.3 UC2: Inserimento dati lavoratori

La Figura 4.2 presenta il diagramma dei casi d'uso relativo all'UC2.

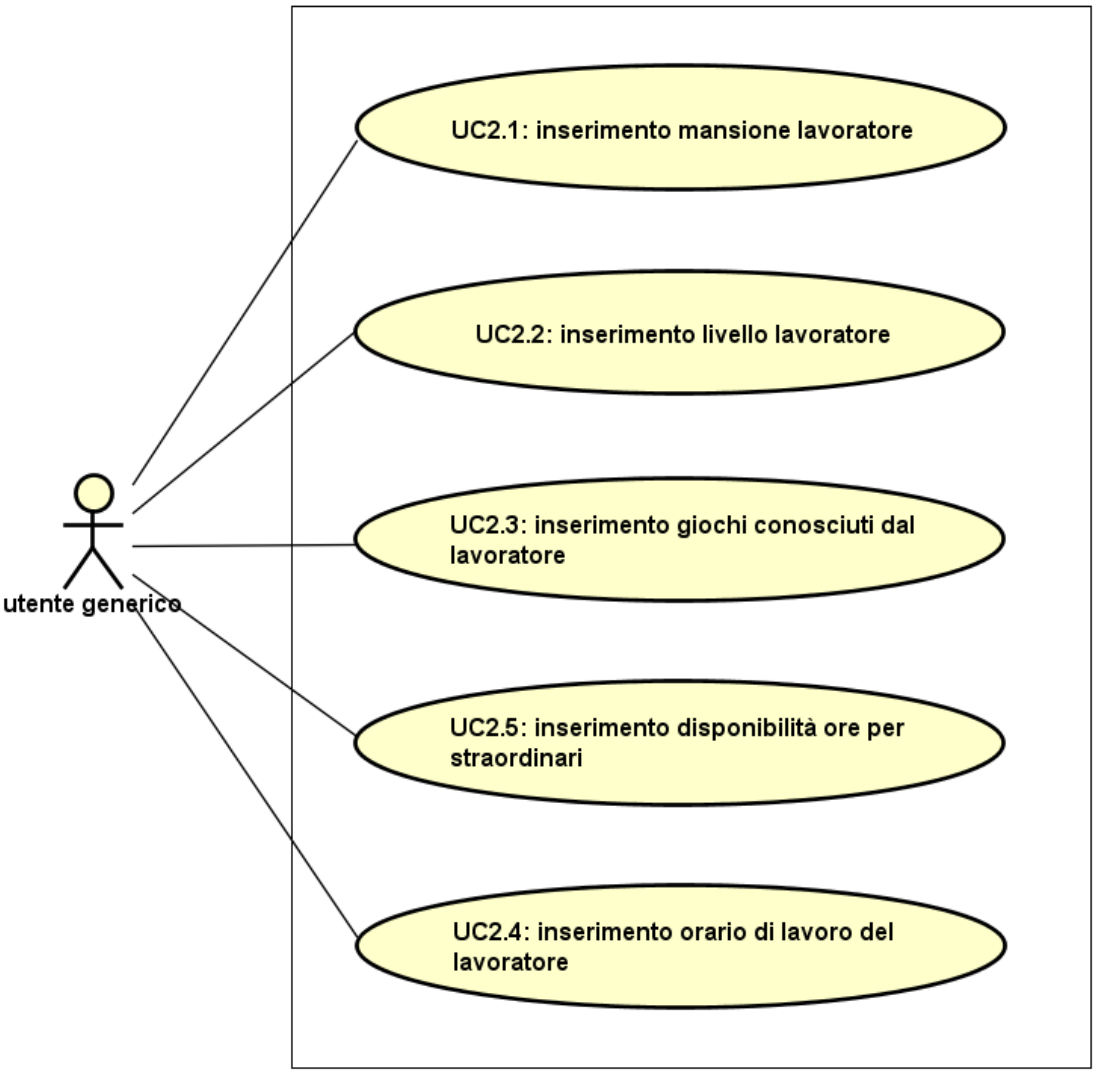


Figura 4.2: UC2: Inserimento dati lavoratori

* **Attori**: utente generico;
* **Descrizione**: l'utente inserisce i dati in input al programma per realizzare uno scheduling riguardanti i lavoratori.

UC2.1: Inserimento mansione lavoratore

* **Attori**: utente generico;
* **Descrizione**: l'utente inserisce i dati in input al programma circa la mansione del lavoratore (dealer, dealer-inspector, inspector).

UC2.2: Inserimento livello lavoratore

* **Attori**: utente generico;
* **Descrizione**: l'utente inserisce i dati in input al programma circa il livello del lavoratore (dealer: 1-8, dealer-inspector: 3, inspector: 1-3).

UC2.3: Inserimento giochi conosciuti dal lavoratore

* **Attori**: utente generico;
* **Descrizione**: l'utente inserisce i dati in input al programma circa i giochi conosciuti dal lavoratore.

UC2.4: Inserimento orario di lavoro del lavoratore

* **Attori**: utente generico;
* **Descrizione**: l'utente inserisce i dati in input al programma circa l'orario di lavoro del giocatore (ora di inizio, ora di fine).

UC2.5: Inserimento disponibilità ore per straordinari

* **Attori**: utente generico;
* **Descrizione**: l'utente inserisce i dati in input al programma circa la disponibilità a svolgere straordinari (ore).

4.1.4 UC3: Inserimento dati postazioni

La Figura 4.3 presenta il diagramma dei casi d'uso relativo all'UC3.

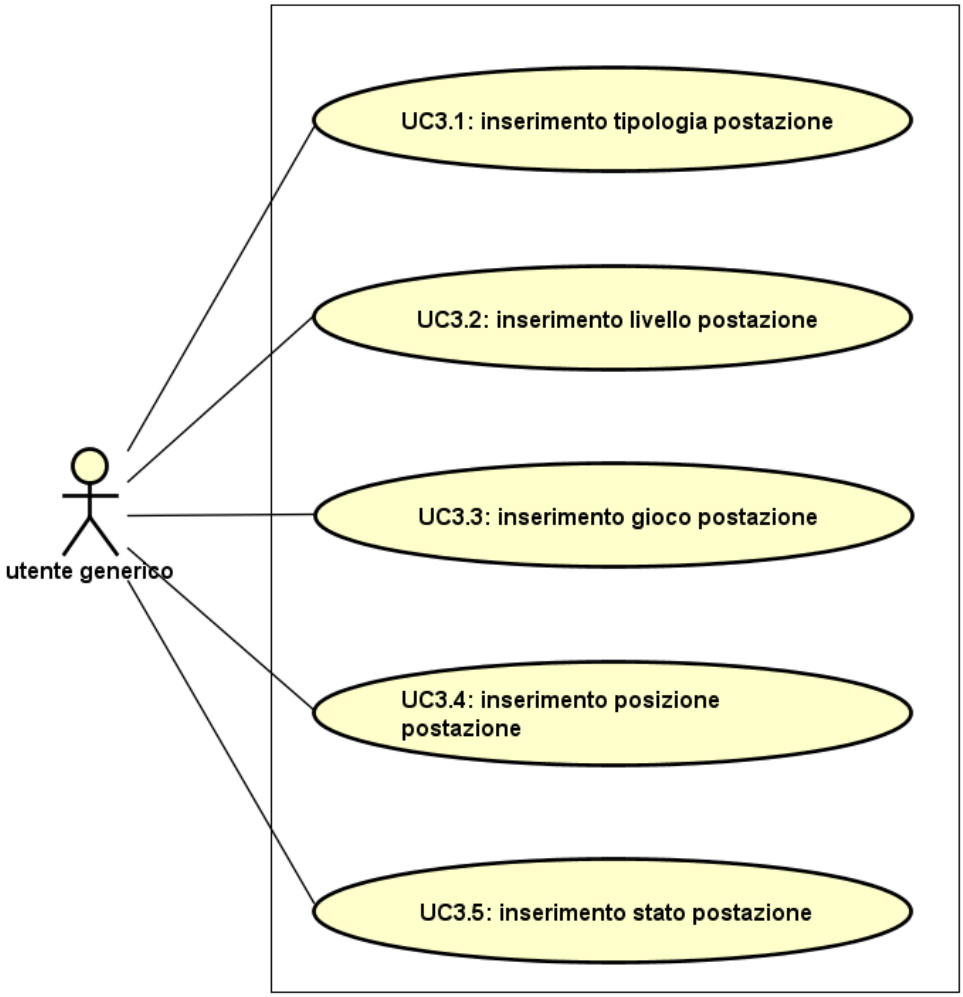


Figura 4.3: UC3: Inserimento dati postazioni

* **Attori**: utente generico;
* **Descrizione**: l'utente inserisce i dati in input al programma per realizzare uno scheduling riguardanti le postazioni.

UC3.1: Inserimento tipologia postazione

* **Attori**: utente generico;
* **Descrizione**: l'utente inserisce i dati in input al programma circa la tipologia della postazione (tavolo, pit).

UC3.2: Inserimento livello postazione

* **Attori**: utente generico;
* **Descrizione**: l'utente inserisce i dati in input al programma circa il livello della postazione (tavoli: 1-8, pit: 1-3).

UC3.3: Inserimento gioco postazione

* **Attori**: utente generico;
* **Descrizione**: l'utente inserisce i dati in input al programma circa il gioco che si gioca nella postazione.

UC3.4: Inserimento posizione postazione

* **Attori**: utente generico;
* **Descrizione**: l'utente inserisce i dati in input al programma circa la posizione che si assume lavorando nella postazione (seduta, in piedi).

UC3.5: Inserimento stato

* **Attori**: utente generico;
* **Descrizione**: l'utente inserisce i dati in input al programma circa lo stato della postazione (aperta, chiusa).

4.1.5 UC4: Inserimento dati turni

La Figura 4.4 presenta il diagramma dei casi d'uso relativo all'UC4.

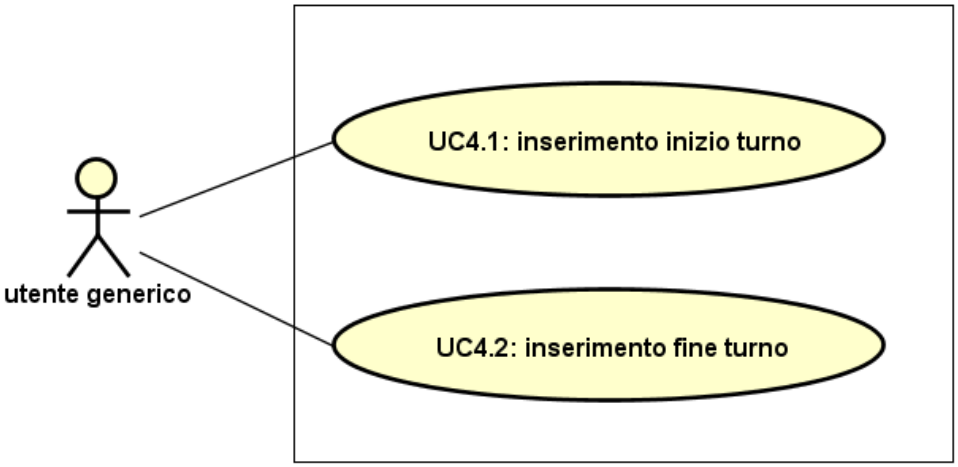


Figura 4.4: UC4: Inserimento dati turni

* **Attori**: utente generico;
* **Descrizione**: l'utente inserisce i dati in input al programma per realizzare uno scheduling riguardanti i turni.

UC4.1: Inserimento inizio turno

* **Attori**: utente generico;
* **Descrizione**: l'utente inserisce i dati in input al programma circa l'ora di inizio di un turno.

UC4.2: Inserimento fine turno

* **Attori**: utente generico;
* **Descrizione**: l'utente inserisce i dati in input al programma circa l'ora di fine di un turno.

4.1.6 UC5: Inserimento data

* **Attori**: utente generico;
* **Descrizione**: l'utente inserisce la data per la quale vuole trovare uno scheduling (importante perché gli orari del personale possono variare di data in data).
* **Scenario alternativo**: se l'utente non inserisce una data, viene trovato lo scheduling per il giorno corrente.

4.1.7 UC6: Inserimento numero turni

* **Attori**: utente generico;
* **Descrizione**: l'utente inserisce il numero di turni per il quale vuole trovare uno scheduling.
* **Scenario alternativo**: se l'utente non inserisce un numero, viene trovato lo scheduling per tutti i turni.

4.1.8 UC7: Visualizzazione dello scheduling in output

* **Attori**: utente generico;
* **Descrizione**: l'utente visualizza l'output del programma (soluzione ammissibile o non).
* **Estensioni**: non c'è output perché è stato commesso un errore di inserimento dei dati.

4.2 Requisiti

Sono riportati di seguito i requisiti individuati per il progetto con rispettivo codice identificativo, importanza e breve descrizione.

* **Codice identificativo:** è la sigla che identifica ogni requisito e rispetta la seguente notazione:

  R[Tipo][Importanza][Identificativo]

  Tipo:

  - **F:** requisito funzionale;
  - **V:** requisito di vincolo;
  - **Q:** requisito qualitativo;

  Importanza:

  - **O:** requisito obbligatorio;
  - **D:** requisito desiderabile;
  - **F:** requisito opzionale.

  Identificativo: codice che identifica in modo univoco un requisito. Viene rappresentato con la notazione decimale puntata per rendere più intuibile la forma gerarchica.

* **Descrizione:** indica testualmente la necessità rappresentata dal requisito.

Nel paragrafo "7.1 - Raggiungimento degli obiettivi" è presente un consuntivo che riporta quanti dei requisiti elencati di seguito sono stati soddisfatti nel corso dello stage.

4.2.1 Requisiti Funzionali

I requisiti funzionali sono indicati nella Tabella 4.1.

Tabella 4.1: Tabella dei requisiti funzionali

ID	Importanza	Descrizione
RFO1	Obbligatorio	L'utente deve poter fornire l'input al programma.
RFO1.1	Obbligatorio	L'utente deve poter fornire l'input circa gli *Item* al programma.
RFO1.2	Obbligatorio	L'utente deve poter fornire l'input circa i *Task* al programma.
RFO1.3	Obbligatorio	L'utente deve poter fornire l'input circa i *TTB* al programma.

ID	Importanza	Descrizione
RFO2	Obbligatorio	Il programma deve poter caricare l'input fornito dall'utente.
RFO2.1	Obbligatorio	Se l'input non è corretto, il programma deve dare un messaggio di errore.
RFO3	Obbligatorio	L'utente deve scegliere la data per cui eseguire lo scheduling.
RFD4	Desiderabile	L'utente deve poter bloccare la produzione dello scheduling dopo un certo numero di turni.
RFO5	Obbligatorio	Il programma deve dedurre da che *TTB* cominciare lo scheduling.
RFO5.1	Obbligatorio	Se l'utente vuole produrre uno scheduling per un giorno diverso dal corrente, lo scheduling va cominciato dal primo *TTB*.
RFO5.2	Obbligatorio	Se l'utente vuole produrre uno scheduling per il giorno corrente e l'orario di inizio del primo *TTB* non è ancora arrivato, lo scheduling va cominciato dal primo *TTB*.
RFO5.3	Obbligatorio	Se l'utente vuole produrre uno scheduling per il giorno corrente e i *TTB* sono già cominciati, lo scheduling va cominciato dal *TTB* successivo al corrente.
RFO6	Obbligatorio	Il programma deve produrre uno scheduling rispettando i vincoli imposti dal problema.
RFO7	Obbligatorio	Il programma deve produrre uno scheduling fair.
RFO8	Obbligatorio	Se non esiste una soluzione ammissibile, il programma deve comunque fornirne una con l'ausilio del Dummy Worker.
RFO9	Obbligatorio	Il programma deve compiere delle iterazioni migliorative e fornire più di uno scheduling.
RFO9.1	Obbligatorio	Le iterazioni sono considerate migliorative se riducono il costo degli *Item*.
RFO10	Obbligatorio	Il programma deve essere flessibile circa la disponibilità dei *Task*.

ID	Importanza	Descrizione
RFO11	Obbligatorio	Il programma deve essere flessibile circa la disponibilità degli *Item*.
RFO12	Obbligatorio	Il programma deve creare un output per l'utente con gli scheduling prodotti.
RFO13	Obbligatorio	Il programma deve creare un output per l'utente con gli scheduling prodotti.
RFF14	Opzionale	Gli output devono avere facilitazioni all'uso per l'utente.
RFF14.1	Opzionale	L'utente deve poter individuare facilmente le assegnazioni dello stesso *Task*.

4.2.2 Requisiti di Qualità

I requisiti di qualità sono indicati nella Tabella 4.2.

Tabella 4.2: Tabella dei requisiti di qualità

ID	Importanza	Descrizione
RQO1	Obbligatorio	Deve essere fornito il manuale dello sviluppatore che descrive classi, metodi e funzionamento del software.
RQO2	Obbligatorio	Deve essere fornito il manuale utente per aiutare l'utente nell'utilizzo dell'applicazione.
RQO3	Obbligatorio	Devono essere fornite relazioni tecniche circa gli studi svolti.
RQO4	Obbligatorio	Devono essere fornite relazioni tecniche circa i risultati ottenuti.
RQD5	Desiderabile	Deve essere fornita una relazione circa l'esplorazione di altri contesti applicativi.

4.2.3 Requisiti di Vincolo

I requisiti di vincolo sono indicati nella Tabella 4.3.

Tabella 4.3: Tabella dei requisiti di vincolo

ID	Importanza	Descrizione
RVO1	Obbligatorio	Il framework per la risoluzione dei problemi di scheduling deve andare ad integrarsi al framework aziendale.
RVO2	Obbligatorio	Il problema di scheduling specifico deve essere risolto utilizzando il framework aziendale esteso.
RVO3	Obbligatorio	L'input e l'output devono esere gestiti tramite file xls.
RVF4	Opzionale	Visualizzazione dell'output tramite HTML.
RVF5	Opzionale	Il modello deve essere implementato in AMPL.

Capitolo 5

Progettazione e codifica

Questo capitolo presenta lo studio della parte di interesse del framework aziendale e l'attività di progettazione svolta per il progetto, approfondita con diagrammi UML. La progettazione viene organizzata in due parti: progettazione dell'estensione del framework e progettazione dell'applicazione, per mantenere separate le due parti di software prodotte durante lo svolgimento dello stage. La progettazione è avvenuta seguendo un approccio top-down: dapprima sono state individuate le componenti ad alto livello e la loro interazione, fino all'individuazione dei singoli moduli e delle classi.

5.1 Premessa: architettura e funzionamento del framework

Il framework aziendale, come già accennato, offre una base per l'implementazione di algoritmi di ottimizzazione e consiste di librerie per l'utilizzo di grafi, di algoritmi euristici di tipo greedy e di meta-euristiche, ad esempio Tabu Search. Per l'estensione del framework con la risoluzione dei problemi di scheduling, durante lo stage ho dovuto utilizzare la parte di libreria atta all'implementazione di algoritmi greedy; di seguito ne viene riportata la sua struttura e ne sono messi in evidenza i metodi di interesse. L'architettura è mostrata in Figura 5.1.

Algorithm la classe `Algorithm` è un'interfaccia per l'implementazione di qualsiasi algoritmo. Ha un unico metodo virtuale puro, `run()`, che deve essere implementato nelle sue classi derivate con il corpo dell'algoritmo di risoluzione; esso deve ritornare un `RunStatus`: SUCCESS_RUN, FAIL_RUN, UNFEASIBLE.

AlgorithmSol è derivata da `Algorithm` ma ancora astratta in quanto non dà un'implementazione del metodo `run()`. Essa rappresenta un ulteriore strato prima di poter implementare gli algoritmi, in quanto contiene un riferimento alla classe `Solution`. Il suo unico metodo `isSolImproving()` serve a confrontare gli objective di due `Solution` per stabilire quale delle due sia la migliore.

Candidate è interna a `Greedy` e rappresenta un candidato da valutare all'interno del metodo `run()`. Va estesa con classi derivate che aggiungano le strutture dati. Una lista di tutti i candidati è campo dati di `Greedy`.

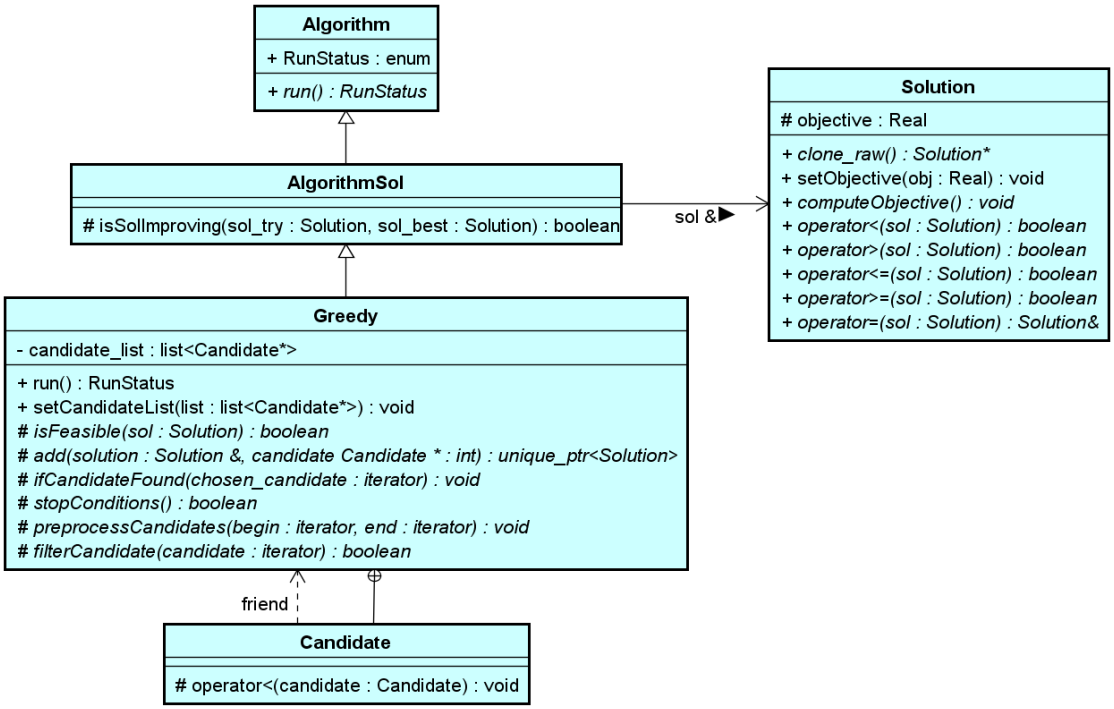


Figura 5.1: Architettura di dettaglio di parte del framework aziendale

Greedy è derivata di AlgorithmSol e rappresenta un template di implementazione per gli algoritmi greedy. Implementa il metodo `run()` con uno "scheletro" generale valido per tutti gli algoritmi di tipo greedy, applicando il *design pattern*[g] "Template Method". Tutti gli altri metodi virtuali rappresentano gli hook per il metodo template e devono essere implementati nelle classi derivate a seconda di che algoritmo greedy si vuole implementare.

Solution è una classe astratta che rappresenta la soluzione trovata all'interno del metodo `run()`. Ha come unico campo dati `objective`, che è il risultato della funzione obiettivo implementata dentro il metodo virtuale puro `computeObjective()`. Nelle classi derivate bisogna implementare tale metodo, gli operatori di confronto, e aggiungere una struttura dati che permetta di mantenere memoria della soluzione.

5.1.1 Il metodo run()

Essendo il punto cardine dell'estensione del framework, è opportuno spiegare il funzionamento del metodo template `run()` implementato all'interno di `Greedy`. A tale scopo se ne riporta uno pseudocodice estremamente semplificato, con i metodi virtuali puri da implementare nelle classi derivate evidenziati in rosso:

```
while(!stopConditions()){
    preprocessCandidates(list_begin, list_end);
    chosen_candidate =list_end;
    list<Candidate*>::iterator candidate;
    for (candidate = list_begin; candidate != list_end; candidate++){
        if (filterCandidate(candidate))
            continue;
        solution_try = add(sol, candidate);
        solution_try.computeObjective();
        if (solution_try < solution_best){
            solution_best = move(solution_try);
            chosen_candidate = candidate;
        }
    }
    setSol(solution_best);
    if (chosen_candidate != list_end)
        ifCandidateFound(chosen_candidate);
    else
        return UNFEASIBLE;
}
return SUCCESS_RUN;
```

Il suo funzionamento è il seguente:

1. Il metodo `stopConditions()` torna `true` quando lo schedule è stato riempito. In tal caso, il metodo run torna `SUCCESS_RUN`. Se il metodo `StopConditions()` torna `false`, lo schedule non è completo e quindi la `run()` continua a ciclare.
2. Il metodo `preprocessCandidates()` permette di modificare per side effect due iteratori `list_begin` e `list_end` che determinano quale set dalla lista di candidati `candidate_list` verranno valutati all'interno del ciclo.
3. Per ogni candidato all'interno del set selezionato:
   (a) Il metodo `filterCandidate()` permette di saltare la valutazione del candidato. Se ritorna `true`, si passa alla valutazione del candidato successivo (punto 3).
   (b) Altrimenti, il metodo `add()` aggiunge a `solution_try` il candidato e ne valuta l'obiettivo.
   (c) Se `solution_try` è migliorante rispetto a `solution_best`, `solution_try` diventa `solution_best` e il candidato corrente diventa `chosen_candidate`.
4. Una volta valutati tutti i candidati, viene impostata `solution_best` come soluzione contenuta in `AlgorithmSol`.
5. Se è stato scelto un candidato, il metodo `ifCandidateFound()` permette di eseguire delle operazioni sul candidato successive alla sua assegnazione. Altrimenti, se nessun candidato è stato scelto, la soluzione non è ammissibile e il metodo torna `UNFEASIBLE`.
6. Torno al punto 1.

5.2 Progettazione dell'estensione del framework

Nella Figura 5.2 viene presentata l'architettura progettata per l'estensione del framework.

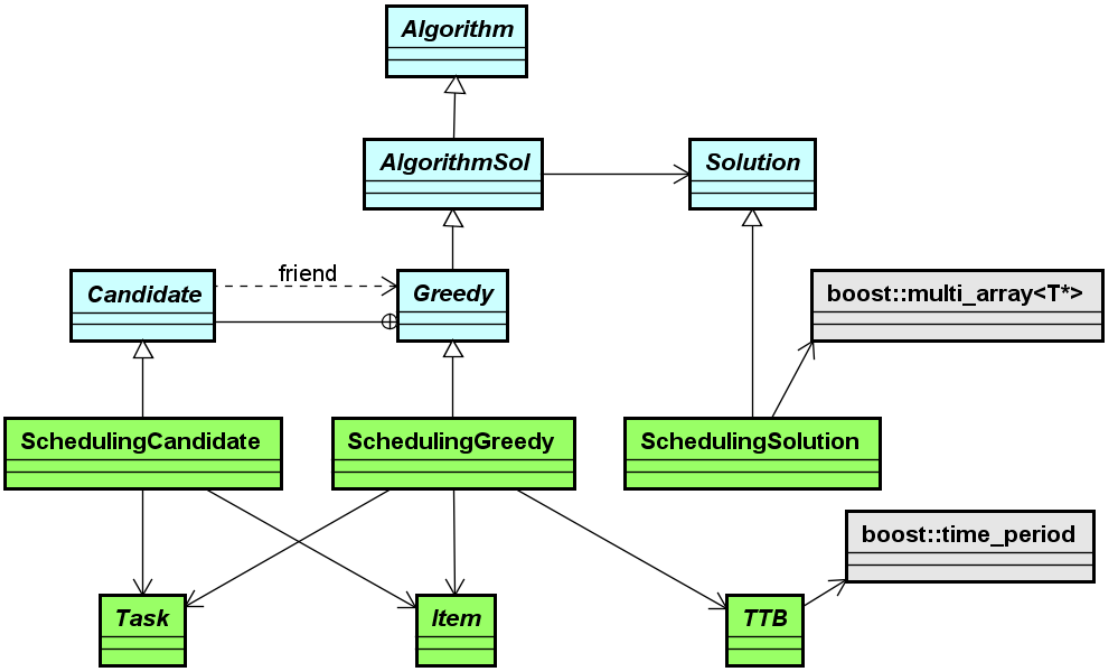


Figura 5.2: Architettura generale dell'estensione al framework

Task classe che modella un generico *Task* di un problema di scheduling e contiene le strutture dati per salvarne le informazioni relative, ad esempio l'essere aperto/chiuso, o la popolarità.

TTB classe che modella un generico *TTB* di un problema di scheduling e contiene le strutture dati per salvarne le informazioni relative, ad esempio gli orari di inizio e fine, o la popolarità.

Item classe astratta che modella un generico *Item* di un problema di scheduling e contiene le strutture dati per salvarne le informazioni relative, ad esempio gli orari di lavoro, la disponibilità a svolgere straordinari, o la popolarità/fairness che acquisisce man mano che che gli vengono assegnati dei *Task*. La classe viene approfondita nel paragrafo 6.1 di questo capitolo, relativo alla progettazione di dettaglio.

SchedulingSolution classe che eredita da `Solution` e definisce, utilizzando la libreria Boost, un `multi_array<Item*,2>` per salvare lo scheduling prodotto dal metodo `run()`. Implementa inoltre il metodo `computeObjective()` calcolando l'obiettivo secondo gli starordinari svolti, la popolarità e la fairness accumulate da tutti gli oggetti di tipo `Item`.

SchedulingGreedy classe che eredita da `Greedy` e ne implementa quindi tutti i metodi virtuali puri. La classe viene approfondita nel paragrafo 6.2 di questo capitolo, relativo alla progettazione di dettaglio.

SchedulingCandidate classe che eredita da `Candidate` e modella i candidati da valutare all'interno del metodo `run()`. Ogni `SchedulingCandidate` è composto da una coppia `Item*`, `Task*` e indica che l'oggetto di tipo `Task` può essere assegnato all'oggetto di tipo `Item`.

5.3 Progettazione dell'applicazione

Nella Figura 5.3 viene presentata l'architettura progettata per la risoluzione del problema del casinò.

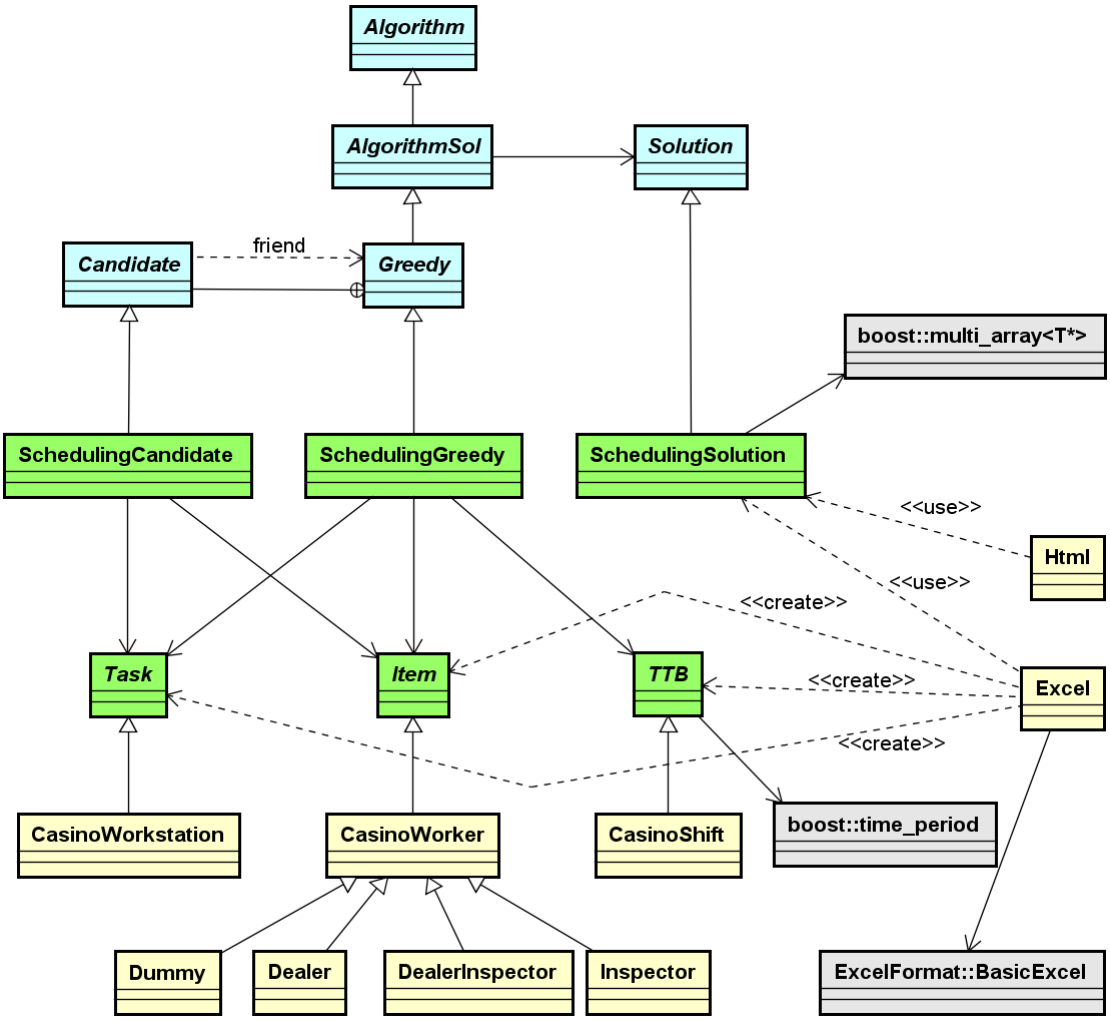


Figura 5.3: Architettura generale dell'applicazione al problema del casinò

CasinoWorkstation classe derivata da `Task`, di cui costituisce una specializzazione. Aggiunge dei campi dati per indicare se la postazione è un tavolo o un pit, il gioco, e il livello.

CasinoShift classe derivata da `TTB`, di cui costituisce una specializzazione. Aggiunge un campo dati per indicare il nome del turno.

CasinoWorker classe derivata da `Item`, di cui costituisce una specializzazione. Aggiunge dei campi dati per indicare il livello del lavoratore e i giochi da lui conosciuti, oltre a implementare i metodi virtuali puri ereditati da `Item`. La classe viene approfondita nel paragrafo 6.1 di questo capitolo, relativo alla progettazione di dettaglio.

Dealer classe derivata da `CasinoWorker` che modella un dealer.

Inspector classe derivata da `CasinoWorker` che modella un inspector.

DealerInspector classe derivata da `CasinoWorker` che modella un dealer-inspector.

Dummy classe derivata da `CasinoWorker` che modella il Dummy Worker.

Excel classe che, utilizzando la libreria esterna `ExcelFormat` si occupa dell'input e output tramite fogli xls.

Html classe che produce l'output in html.

5.4 Librerie

5.4.1 ExcelFormat

ExcelFormat [5] è una libreria open source che permette di leggere, scrivere e modificare file *XLS*[g] tramite C++. La scelta è ricaduta sull'utilizzare questa libreria per gestire l'input e output in quanto rappresentava un compromesso fra rapidità di programmazione (dato che l'interfaccia utente, come già detto, non era ritenuta prioritaria) e un'interfaccia con un certo grado di controllo dell'input, grazie alle funzioni di Excel, e quanto meno usabile lato utente per inserire gli input e visionare gli output.
ExcelFormat offre funzioni per maneggiare i file XLS e i diversi worksheet presenti al loro interno, leggere e scrivere il contenuto delle celle, modificarne la formattazione. Tuttavia le funzioni che offre sono basilari, ad esempio non è possibile scrivere formule.

5.4.2 Boost

Boost è una collezione di librerie open source che estendono le funzionalità del C++. In particolare, all'interno del progetto sono state usate le librerie:

* **date_time:** per gestire orari e date. È stata preferita a *std::time* in quanto offre più funzionalità ed è di utilizzo più immediato.
* **multi_array:** per gestire array dinamici multidimensionali. È stata preferita ad annidamenti di contenitori della libreria standard, come *std::vector*, in quanto ne offre un'implementazione più efficiente.

5.5 Design pattern

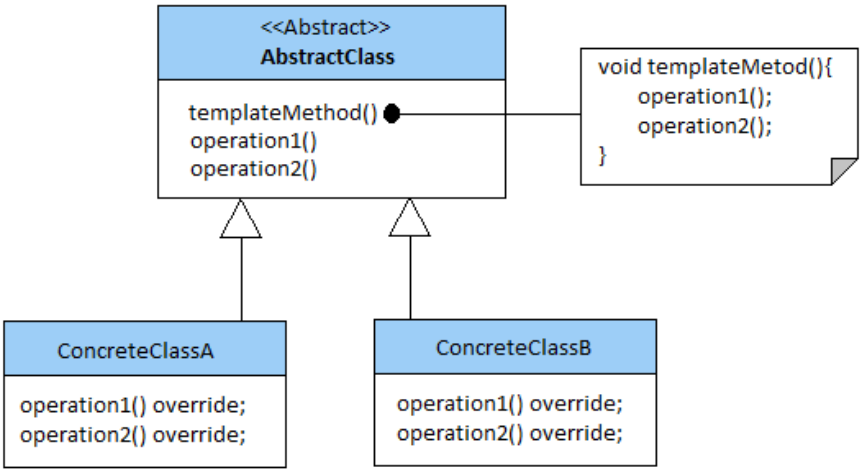


Figura 5.4: Template Method

Template Method Il Template Method, la cui architettura è presentata nella Figura 5.4, è un design pattern comportamentale che permette di definire lo scheletro di un algoritmo lasciando alle sottoclassi il compito di implementarne alcuni passi come preferiscono. In questo modo si può ridefinire e personalizzare parte del comportamento nelle varie sottoclassi senza dover riscrivere più volte il codice in comune e senza alterarne la struttura. Tale design pattern è stato descritto per la prima volta in [8].

5.6 Progettazione di dettaglio

In questa sezione viene approfondita, tramite diagrammi UML e descrizione dettagliata di classi, metodi e attributi, la progettazione di dettaglio delle classi principali del progetto. Vengono tralasciate le classi più semplici o meno importanti.

5.6.1 Gerarchia di Item

Nella Figura 5.5 viene presentata la progettazione di dettaglio della gerarchia con classe base `Item`.

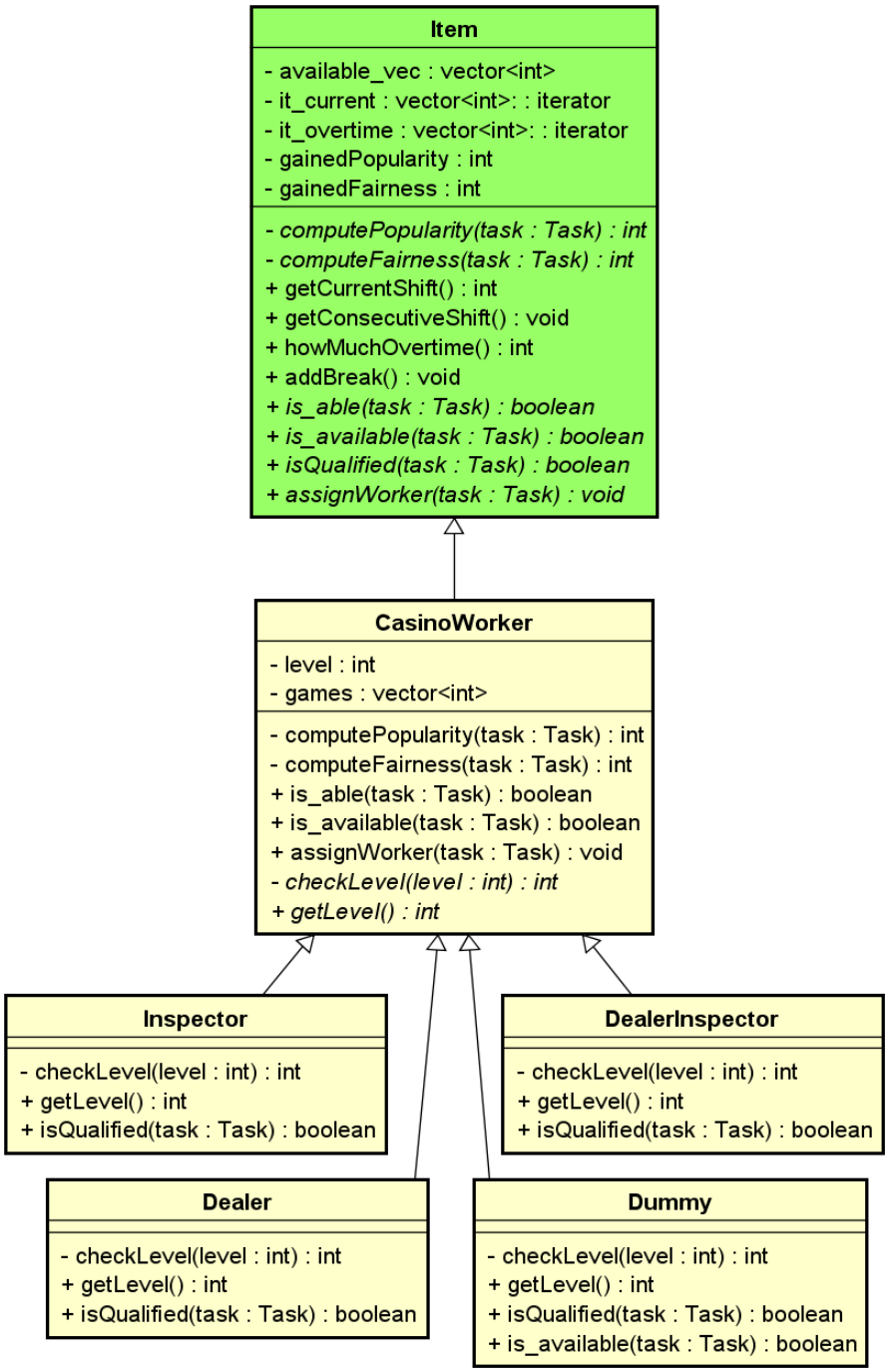


Figura 5.5: Gerarchia della classe Item

Item

* **Attributi**
  - `available_vec` vettore la cui dimensione è uguale al numero di *TTB*. Inizialmente ogni elemento segnala se l'*Item* è disponibile per il *TTB* (valore -1) o no (valore -2); man mano che l'*Item* viene assegnato ai *Task* durante i *TTB*, le celle vengono riempite con gli id dei *Task* svolti.
  - `it_current` iteratore per available_vec, accede all'elemento corrispondente al corrente *TTB*.
  - `it_overtime` iteratore per available_vec, accede all'elemento corrispondente al *TTB* in cui l'*Item* comincia a lavorare gli straordinari.
  - `gainedPopularity` tiene memoria della popolarità accumulata dall'*Item*.
  - `gainedFairness` tiene memoria della fairness accumulata dall'*Item*.
* **Metodi**
  - `getCurrentShift` ritorna l'id del *TTB* corrente.
  - `getConsecutiveShift` calcola e ritorna quanti *TTB* consecutivi (senza pause) ha lavorato l'*Item*.
  - `howMuchOvertime` calcola e ritorna quanti *TTB* di straordinari ha lavorato l'*Item*.
  - `addBreak` mette in pausa l'*Item* per il *TTB* corrente (lascia it_current a -1 e lo fa avanzare di uno).
* **Metodi virtuali puri**
  - `computePopularity` per calcolare la popolarità accumulata da ogni *Item*.
  - `computeFairness` per calcolare la fairness accumulata da ogni *Item*.
  - `is_able` verifica se un *Item* è capace di lavorare ad un certo *Task*.
  - `is_available` verifica se un *Item* è disponibile per lavorare ad un certo *Task*.
  - `isQualified` verifica che la mansione di un *Item* sia compatibile con quella richiesta per un *Task*.
  - `assignWorker` per eseguire le operazioni che seguono un assegnamento sull'*Item*.

CasinoWorker

* **Attributi**
  - `level` è il livello del lavoratore.
  - `games` vettore dei giochi conosciuti dal lavoratore.
* **Metodi**
  - `computePopularity` calcola la popolarità accumulata dal lavoratore, tenendo in conto il livello delle postazioni in cui il lavoratore è stato assegnato, quanti turni senza pause ha fatto, quali turni ha fatto e dove.

  - `computeFairness` calcola la fairness accumulata dal lavoratore, tenendo in conto se ha lavorato in postazioni dove si siede o si sta in piedi.
  - `is_able` verifica se il lavoratore è in grado di lavorare in una certa postazione o no, in base alla sua conoscenza del gioco, al livello, alla mansione.
  - `is_available` verifica se il lavoratore è disponibile a lavorare in una certa postazione, in base non solo alle sue capacità, ma anche dall'essere presente in sala, non dover andare in pausa, non dover rotare di postazione.
  - `assignWorker` esegue le operazioni che seguono l'assegnamento del lavoratore: aggiorna available_vec e fa avanzare it_current, calcola gainedPopularity e gainedFairness.

* **Metodi virtuali puri**
  - `checkLevel` esegue una verifica dell'input riguardante il livello.
  - `getLevel` ritorna il livello del lavoratore.

Dealer

* **Metodi**
  - `checkLevel` controlla che l'input riguardante il livello sia compreso fra 1 e 8. Se sì, ritorna il livello incrementato di due.
  - `getLevel` ritorna il livello decrementato di due.
  - `isQualified` ritorna true se la postazione è un tavolo, false altrimenti.

Inspector

* **Metodi**
  - `checkLevel` controlla che l'input riguardante il livello sia compreso fra 1 e 3. Se sì, ritorna il livello.
  - `getLevel` ritorna il livello.
  - `isQualified` ritorna true se la postazione è un pit, false altrimenti.

DealerInspector

* **Metodi**
  - `checkLevel` controlla che l'input riguardante il livello sia 3. Ritorna sempre 3.
  - `getLevel` ritorna 3.
  - `isQualified` ritorna true sia se la postazione è un tavolo, sia se la postazione è un pit.

Dummy

* **Metodi**
  - `checkLevel` Ritorna sempre 1.
  - `getLevel` ritorna 1.
  - `isQualified` ritorna sempre true.
  - `is_available` ritorna sempre true.

5.6.2 SchedulingGreedy

Nella Figura 5.6 viene presentata la progettazione di dettaglio della gerarchia con classe base `Greedy`.

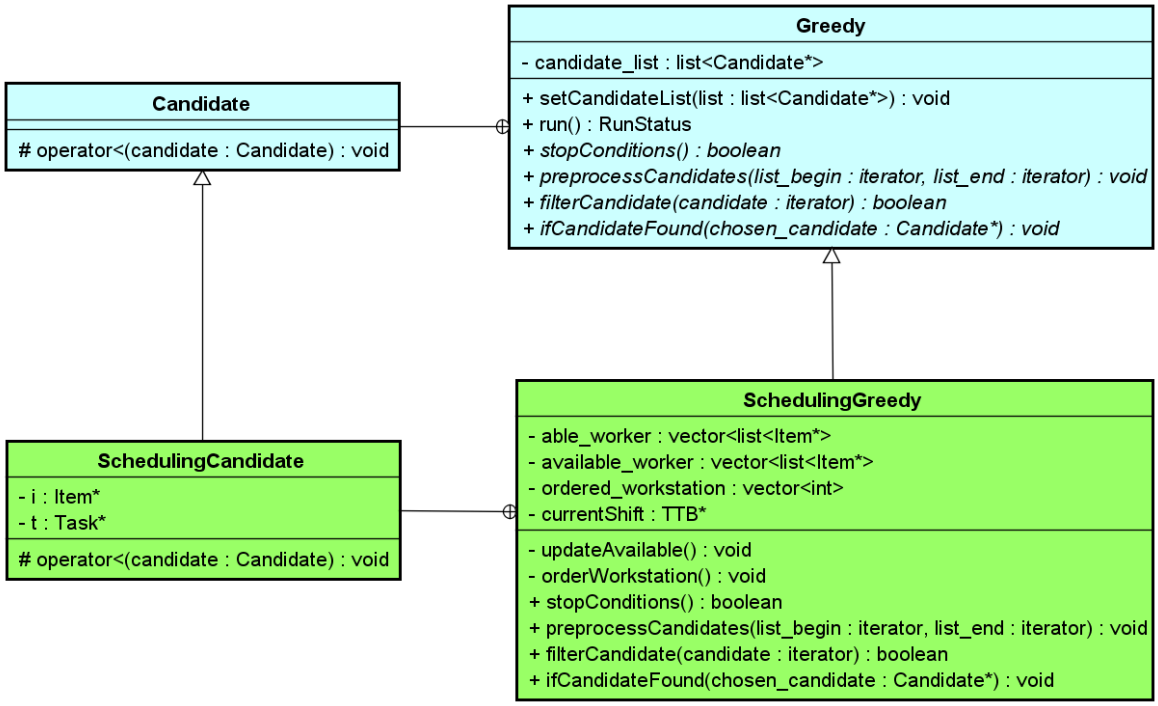


Figura 5.6: Gerarchia della classe Greedy

* **Attributi**
  - `able_worker` vettore di liste, ogni elemento del vettore rappresenta un *Task* e i nodi della lista a cui punta sono gli *Item* capaci di svolgere il *Task* (funzione is_able). Viene costruito una sola volta e costituisce le coppie *Item*-*Task* con cui vengono costruiti i candidati.
  - `available_worker` vettore di liste, ogni elemento del vettore rappresenta un *Task* e i nodi della lista a cui punta sono gli *Item* disponibili a svolgere il *Task* (funzione is_available).
  - `ordered_workstation` vettore dei *Task* ordinati per priorità.
  - `currentShift` *TTB* per il quale l'algoritmo sta schedulando.
* **Metodi**
  - `updateAvailable` aggiorna available_worker, eliminando gli *Item* non più disponibili.
  - `orderWorkstation` ordina il vettore ordered_workstation tramite un counting sort basandosi sul numero di *Item* disponibili per il *Task* (vettore available_worker). Il *Task* con meno *Item* disponibili occupa la posizione 0, in fondo al vettore vengono posizionati i *Task* già assegnati.

- `stopConditions` controlla se SchedulingSolution è stata riempita completamente.
- `preprocessCandidates` fa un update di available_worker, riordina ordered_workstation e poi, basandosi sul *Task* da schedulare come successivo (il primo del vettore) individua il set di candidati da valutare all'interno di candidate_list.
- `filterCandidate` controlla che l'*Item* candidato sia disponibile per il *Task* (available_workers).
- `ifCandidateFound` chiama assignWorker() sull'*Item* scelto per la schedulazione.

5.7 Codifica e sofware realizzato

L'implementazione in C++ dell'architettura sopra descritta ha prodotto il software desiderato secondo le specifiche fornitemi.
Il tutto è stato completato con la scrittura di alcune funzioni in Javascript per rendere più usabile lo scheduling in html fornito in output. In particolare, sono state rese evidenziabili le postazioni all'interno dello scheduling facendoci click sopra; inoltre la prima riga (contente i turni) e la prima colonna (contenente i lavoratori) della tabella dello scheduling sono state rese fisse, in quanto necessariamente la tabella è molto grande e serve quindi uno scroll sia orizzontale che verticale per essere visualizzata per intero.

Capitolo 6

Verifica e validazione

In questo capitolo viene approfondito il periodo di verifica e test, durante il quale sono emerse alcune difficoltà che hanno richiesto un tempo maggiore del previsto per la risoluzione. Viene qui presentata la modalità di risoluzione di queste difficoltà, unita ai risultati ottenuti durante i test e i dati emersi dalle analisi dei risultati.

6.1 Generazione degli input

Una volta accertato che il comportamento della *business logic*[g] del software prodotto fosse corretto su piccoli input di prova, si è presentato il problema di testare l'applicazione su istanze di input più grandi e con più variabili. Da piano di lavoro, il tempo previsto per verifica, testing e analisi dei risultati era stabilito di 20 ore; tuttavia, è stato da subito chiaro che il processo di verifica sarebbe stato più complicato del previsto.
Per eseguire test significativi in serie, era infatti necessario poter produrre automaticamente degli input su cui il software potesse elaborare per produrre schedule: è stata quindi estesa la classe `Excel` per poter andare a modificare in scrittura i fogli xls di input. In un primo momento, si è provato a generare degli input randomici, ma tale soluzione si è rivelata problematica per due motivi: in primo luogo, gli input randomici non producevano praticamente mai delle soluzioni ammissibili in quanto spesso ci si trovava con un sbilanciamento postazioni/lavoratori; in secondo luogo, anche quando veniva (raramente) prodotta una soluzione ammissibile, tale soluzione non poteva essere rappresentativa di un buono scheduling in quanto non era sensata. Il problema principale da risolvere è quindi diventato *come* generare degli input il più possibile vicini alla realtà che potessero produrre degli schedule da analizzare.
Il problema è stato esposto al tutor aziendale, che ha approvato la modifica al piano di lavoro assegnando più ore alla verifica e al testing, tagliando su attività per il raggiungimento di requisiti desiderabili e opzionali, ritenendo più importante poter testare il software in maniera corretta e approfondita.

Secondo le informazioni fornite dal casinò, gli input per essere realistici dovevano:

- comprendere circa 30 postazioni aperte contemporaneamente, per un totale di 40 lavoratori attivi. Il totale dei lavori, per garantire quindi uno schedule fair e rispettoso delle pause e dei vincoli previsti, doveva ammontare a circa il 33% in più rispetto al numero di postazioni.

* di tutte le postazioni aperte, alla maggior parte viene assegnato un livello basso e giochi comunemente conosciuti dai lavoratori, mentre le postazioni che richiedono un livello alto o giochi poco conosciuti sono meno.
* il livello dei lavoratori dipende dal tempo passato all'interno del casinò e dalle loro abilità, di conseguenze la maggior parte dei lavoratori è di livello intermedio, mentre sono pochi quelli di livello molto alto e molto basso.
* i giochi conosciuti dai lavoratori dipende dal loro livello: più un dealer è di livello alto, più giochi conosce; per quanto riguarda inspector e dealer-inspector, essi conoscono praticamente tutti i giochi.

Sulle basi di queste informazioni, è stato possibile produrre un modello probabilistico che descrivesse una situazione reale e progettare una nuova classe, `Test`, che generasse automaticamente ad ogni *run* dell'algoritmo un set di input secondo il modello definito.

6.1.1 Modello probabilistico

Tutti i grafici presenti in questa sezione sono stati prodotti sul sito *Wolphram Alpha* [19].

Numero di postazioni Ipotizzando che il numero di postazioni aperte contemporaneamente sia circa 30, genero il numero di postazioni tramite una distribuzione normale $\mathcal{N}(\mu; \sigma^2)$ con:

* tavoli: $\mu = 20$, $\sigma^2 = \frac{\mu}{5}$
* pit: $\frac{tavoli}{3}$

La Figura 6.1 presenta i grafici per le distribuzioni del numero di postazioni.

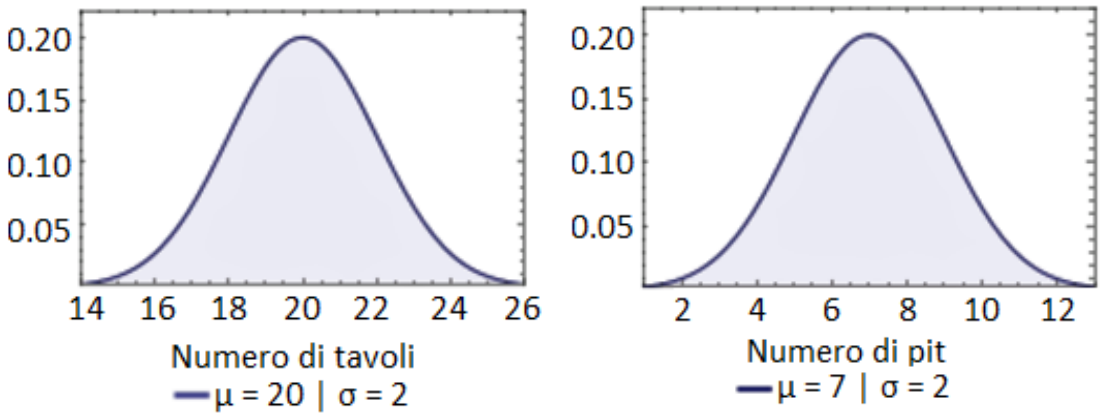


Figura 6.1: Distribuzioni normali per il numero di postazioni

Costruzione delle postazioni Ipotizzando che le postazioni che richiedono un livello alto sono poche e viceversa, genero i livelli delle postazioni L_t con una distribuzione discreta uniforme $\mathcal{U}(\mathcal{S})$ dove

$$\mathcal{S} = \{\alpha + i\,\beta \text{ t.c. } i \in \{0, 2, ..., n\}\}$$

ovvero i cui elementi sono i progressione aritmetica con $\alpha = 1$, $\beta = 1$ e:

* per i tavoli (8 livelli): $n = 7$
* per i pit (3 livelli): $n = 2$

La Figura 6.2 presenta i grafici per le distribuzioni dei livelli delle postazioni.

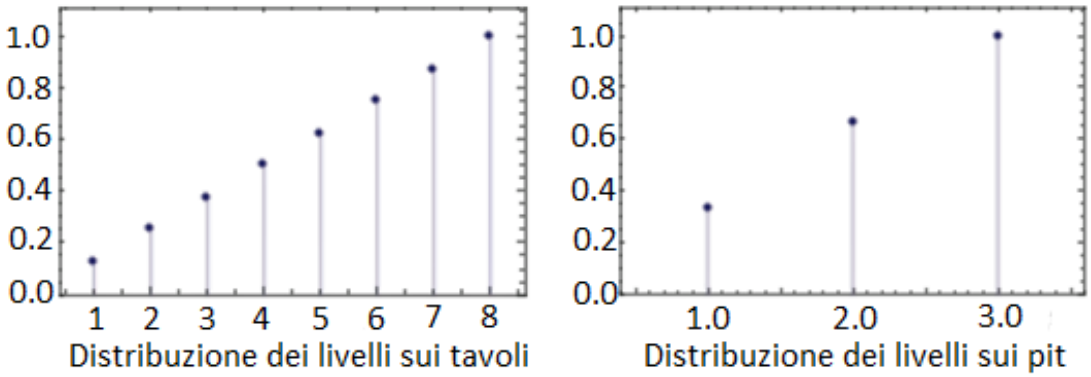


Figura 6.2: Distribuzione discreta uniforme per i livelli delle postazioni

Inoltre viene generato il gioco richiesto ad una certa postazione in modo che la frequenza del gioco base (//) sia doppia rispetto a quella di altri giochi (BJ e PG).
Ad ogni postazione è richiesto un solo gioco.
La Figura 6.3 presenta il grafico delle frequenza di estrazione dei giochi.

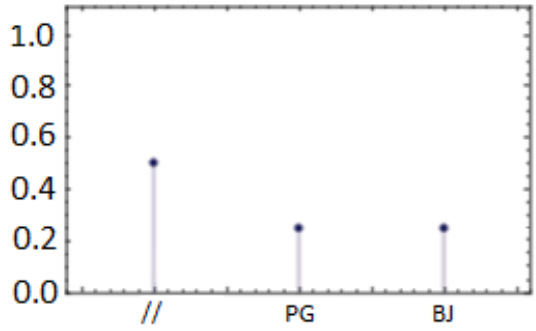


Figura 6.3: Frequenza di estrazione dei giochi per ogni postazione

Numero di lavoratori Ipotizzando che il numero di lavoratori attivi contemporaneamente sia di circa il 33% in più rispetto al numero di postazioni, genero il numero di lavoratori tramite una distribuzione normale $\mathcal{N}(\mu; \sigma^2)$ con:

* dealer: $\mu = 1.6\, tavoli$, $\sigma^2 = \frac{tavoli}{5}$
* inspector: $\mu = 1.6\, pit$, $\sigma^2 = \frac{pit}{5}$

La Figura 6.4 presenta i grafici per le distribuzioni del numero dei lavoratori.

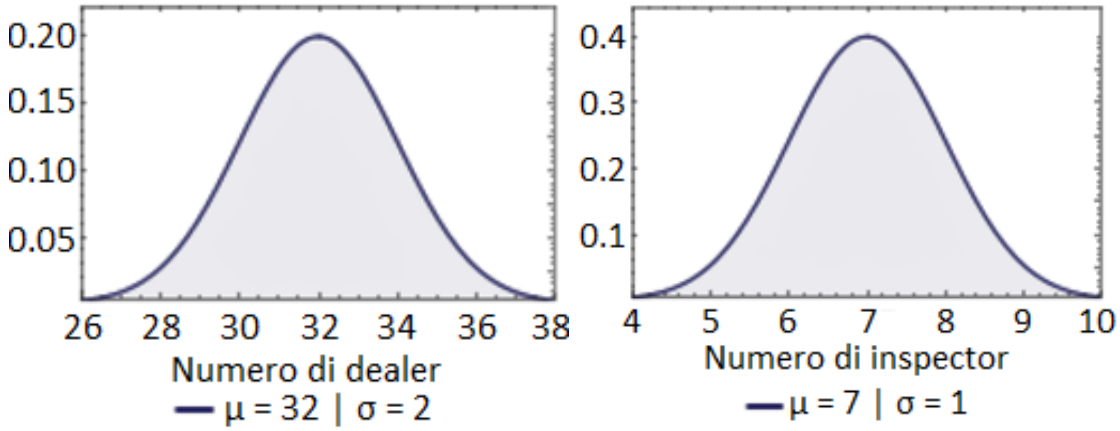


Figura 6.4: Distribuzioni normali per il numero di lavoratori

Da notare che il valor medio μ è stato fissato al 60% in più rispetto al numero di postazioni, sarà poi compito dell'algoritmo andare ad ottimizzare il numero di lavoratori richiesti per riempire lo scheduling.

Per quanto riguarda il numero di dealer-inspector, esso viene fissato a

$$(tavoli + pit)\, 1.6 - dealer - inspector$$

per far sì che il numero di lavoratori attivi sia sempre esattamente il 60% in più rispetto al numero di postazioni.

Costruzione dei dati sui lavoratori Ipotizzando che il livello dei lavoratori dipenda solo dal tempo che hanno passato all'interno del casinò, generiamo i livelli dei lavoratori L_w tramite una distribuzione normale $\mathcal{N}(\mu; \sigma^2)$ con:

* dealer: $\mu = 4.5$, $\sigma^2 = \frac{\mu}{3.5}$
* inspector: $\mu = 2$, $\sigma^2 = \frac{\mu}{1}$

La Figura 6.5 presenta i grafici per le distribuzioni dei livelli dei lavoratori.

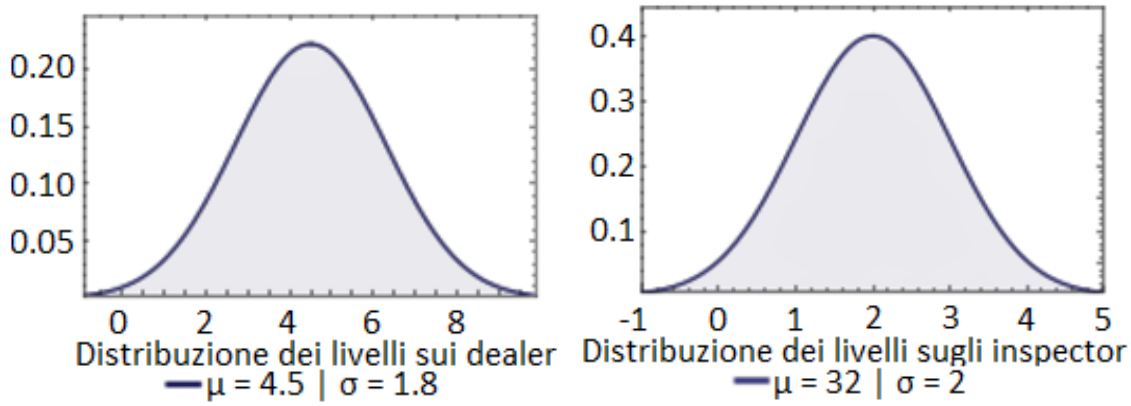


Figura 6.5: Distribuzioni normali per i livelli dei lavoratori

Ipotizzando che la probabilità che un lavoratore conosca un gioco sia direttamente proporzionale al suo livello (più un lavoratore ha livello alto, più giochi conosce; tutti conoscono i giochi base //), generiamo i giochi che ogni lavoratore conosce con una distribuzione di Bernoulli $\mathcal{B}e(p)$ dove $p = 1 - \frac{livello}{10}$.
La Figura 6.6 presenta i grafici per le distribuzioni dei giochi sui lavoratori.

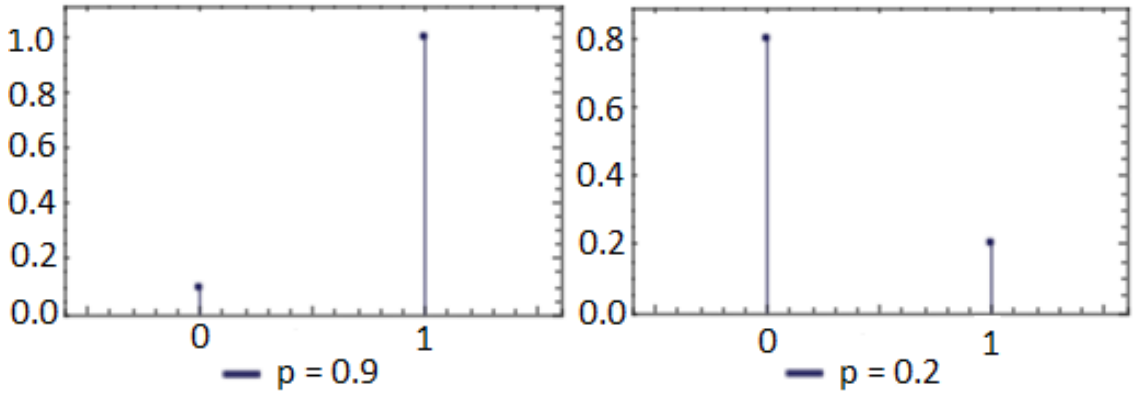


Figura 6.6: Probabilità che due lavoratori di livelli 1 (a sinistra) e 8 (a destra) conoscano un gioco (1) o meno (0)

6.2 Testing

6.2.1 Costruzione dei test

Per eseguire i test in serie, è stato prodotto uno script Python che esegue trenta volta il programma. Ogni esecuzione genera una nuova istanza di input, costruisce gli oggetti, esegue l'algoritmo greedy producendo una soluzione e poi procede con l'ottimizzare il numero di lavoratori necessari al riempimento dello schedule.
Lo script Python tiene traccia, per ogni esecuzione, del tempo impiegato e di tutti gli input e output per poter poi elaborare i dati creando tre tipi di grafici:

* Executions;
* Tests;
* Exec n.

Le tre tipologie vengono presentate nei prossimi paragrafi.

Grafico "Executions" Grafico che mette in relazione il tempo di esecuzione con la grandezza iniziale dell'input (numero tavoli, numero di lavoratori) e di quanto, tramite le iterazioni miglioranti eseguite dal software, si riesce a ridurre il numero di lavoratori. Viene messa in evidenza la media di tutti i valori.

Ad esempio, la Figura 6.7 presenta il grafico "Executions" prodotto su trenta esecuzioni dell'algoritmo.
Il tempo di esecuzione medio è 5.387s, il numero medio di postazioni è 27.5, il numero medio lavoratori all'inizio è di circa 46 (+68.3% rispetto al numero di postazioni) mentre, dopo le iterazioni miglioranti, il numero medio di lavoratori è di circa 37 (+35.5% rispetto al numero di tavoli, valore che ha un riscontro nel numero effettivo di lavoratori nel casinò di Londra).

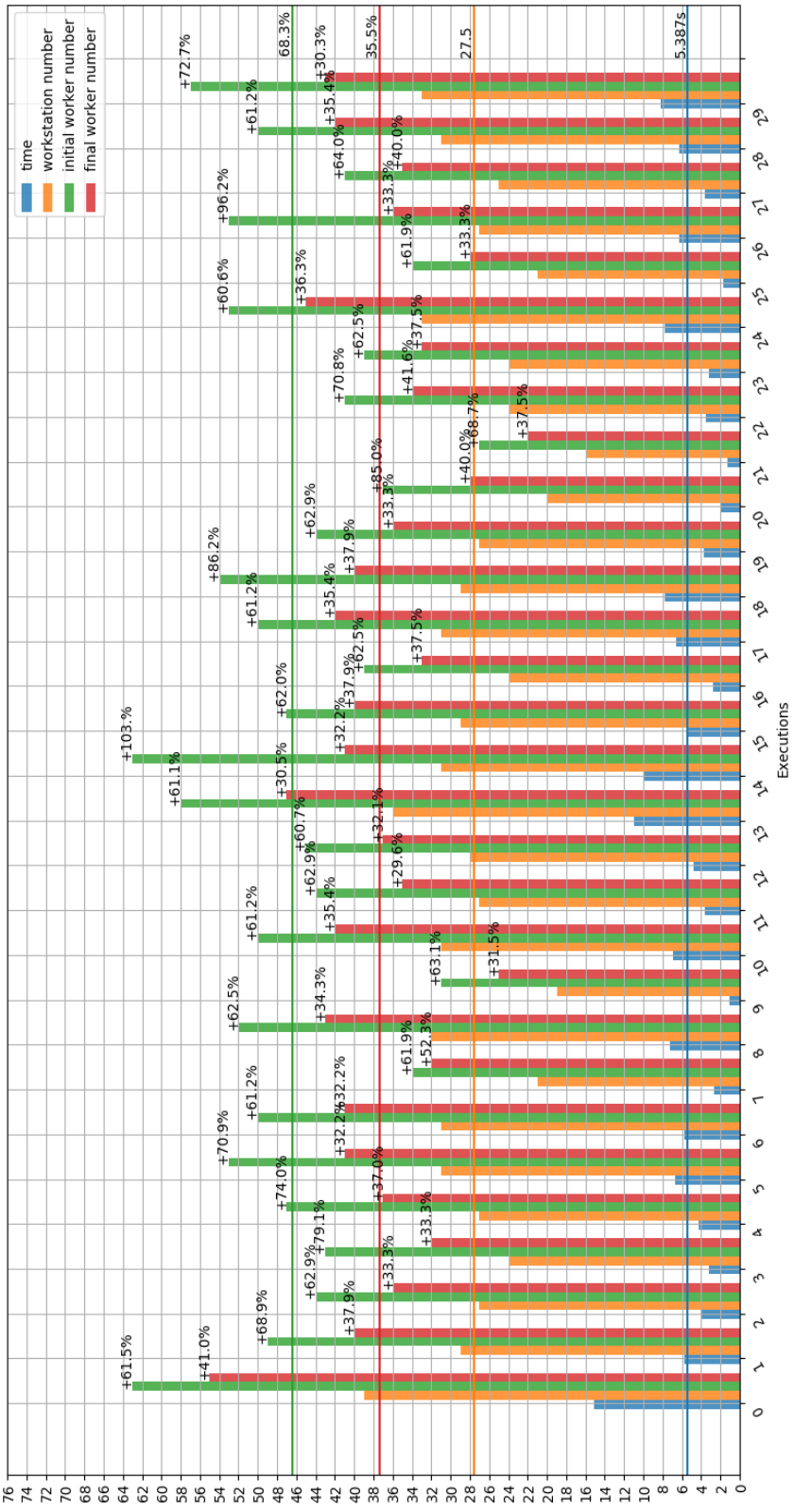


Figura 6.7: Grafico prodotto dai test: Executions

Grafici "Tests" Sono tre grafici diversi che mostrano l'attuazione del modello probabilistico negli input generati dal programma.

Il primo mostra la distribuzione dei livelli sulle postazioni e sui lavoratori. Ricordando che la distribuzione dei livelli sulle postazioni deve seguire la distribuzione discreta uniforme, mentre la distribuzione dei livelli sui lavoratori deve seguire la distribuzione normale (gaussiana), su trenta esecuzioni dell'algoritmo sono state prodotte le distribuzioni visibili in Figura 6.8, che confermano il modello probabilistico:

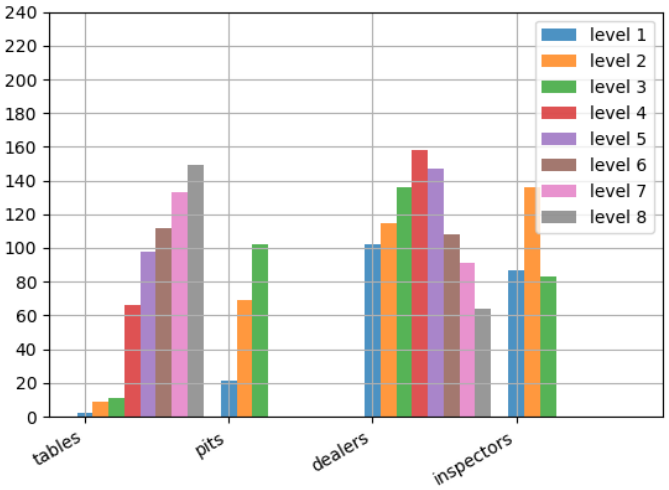


Figura 6.8: Grafico prodotto dai test: Tests (livelli)

Il secondo grafico mostra la distribuzione della conoscenza dei giochi sulle postazioni e sui lavoratori (limitandosi ai dealer, in quanto dealer-inspector e inspector conoscono tutti i giochi). Ricordando che la frequenza dei giochi deve essere il 50% per il gioco base // e il 25% per BJ e PG, su trenta esecuzioni dell'algoritmo sono state prodotte le distribuzioni visibili in Figura 6.9, che confermano il modello probabilistico:

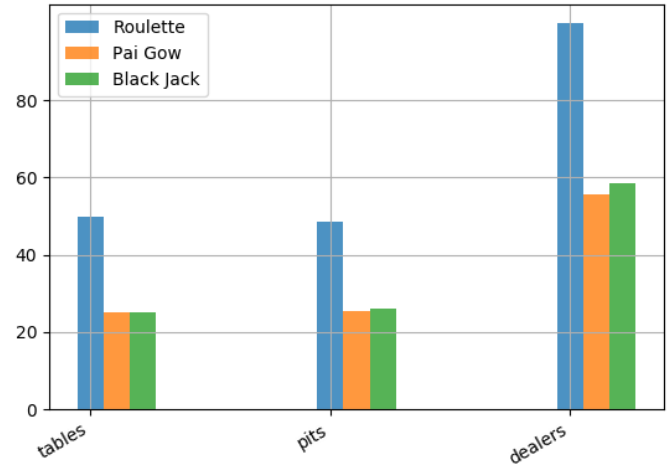


Figura 6.9: Grafico prodotto dai test: Tests (giochi)

Il terzo e ultimo grafico mostra la distribuzione della conoscenza dei giochi sui dealer, suddivisi per livello. Ricordando che la probabilità che un dealer conosca un gioco è direttamente proporzionale al suo livello secondo una distribuzione di Bernoulli (quindi, un livello 1 ha probabilità del 90% di conoscere un gioco, un livello 2 probabilità 80%, e così via fino ad arrivare a un livello 8 che ha probabiblità del 20%), su trenta esecuzioni dell'algoritmo sono state prodotte le distribuzioni visibili in Figura 6.10, che confermano il modello probabilistico:

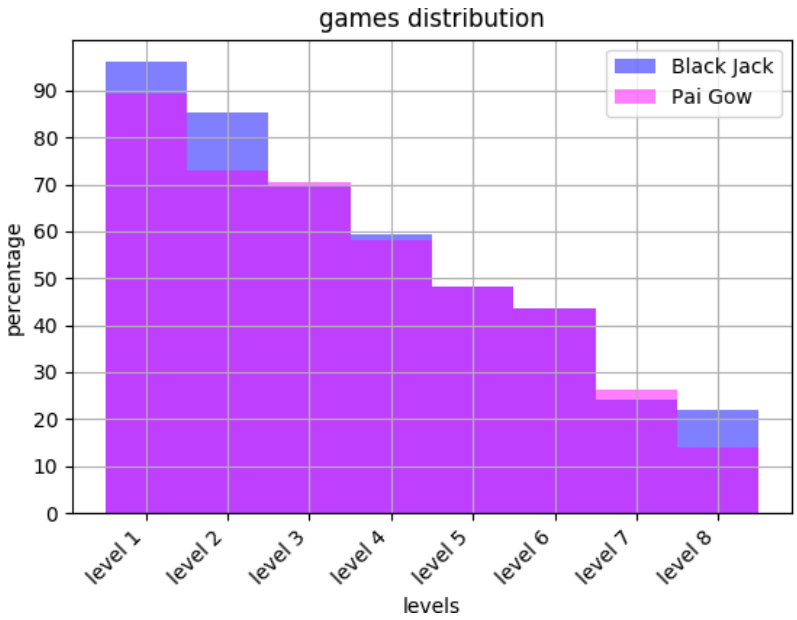


Figura 6.10: Grafico prodotto dai test: Tests (giochi per livelli)

Grafici "Exec n" Infine per ogni esecuzione viene creato un grafico (quindi in tutto, per un'esecuzione dello script, vengono creati trenta grafici) per mostrare la bontà degli input prodotti; essi mostrano il numero di lavoratori prodotti dal modello probabilistico capaci di lavorare in ogni postazione e possono essere di due colori: rosso o verde.
I grafici rossi indicano che i lavoratori prodotti come input per l'esecuzione n non erano ben distribuiti sui tavoli; l'algoritmo non ha quindi trovato una soluzione ammissibile e ha dovuto inserire nello schedule il Dummy Worker.
I grafici verdi, al contrario, indicano che l'algoritmo ha trovato una soluzione ammissibile.
Assumendo che l'algoritmo greedy di riempimento dello scheduling funzioni correttamente, ovvero che trovi una soluzione ammissibile quando ce ne è una, tali grafici diventano una misura della bontà dell'input prodotta dal modello probabilistico. Nella Figura 6.11 si possono vedere le due tipologie di grafici; si noti come nel grafico rosso a sinistra ci siano delle postazioni con un numero molto basso di lavoratori disponibili, mentre nel grafico verde a destra i lavoratori siano meglio distribuiti sulle postazioni.

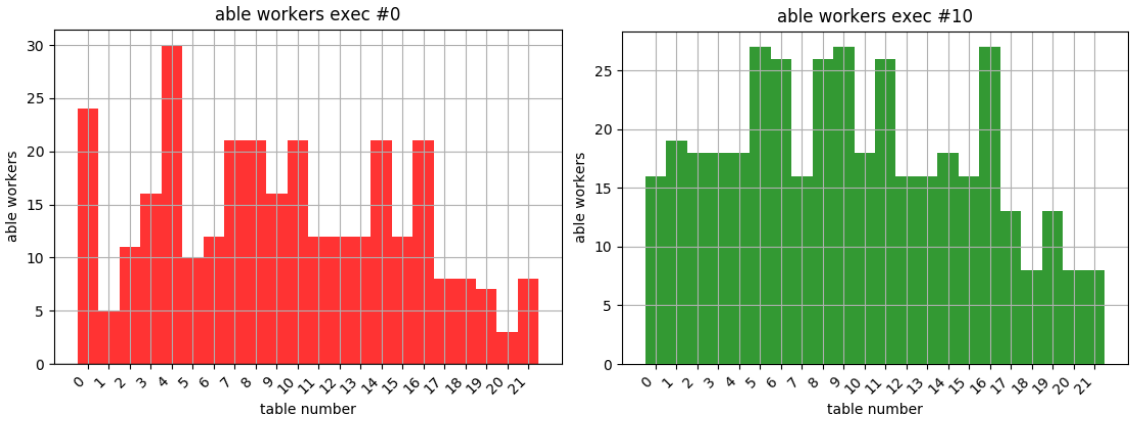


Figura 6.11: Grafici prodotti dai test: Exec n

6.2.2 Analisi dei risultati

Sono stati eseguiti tre test prestazionali modificando lievemente i parametri del modello di generazione degli input, al fine di provare a rispecchiare diversi tipi di situazioni che possono presentarsi all'interno del casinò e per analizzare le reazioni dell'algoritmo.

Primo test prestazionale

Il test viene eseguito sulla distribuzione uniforme dei livelli delle postazioni, tuttavia limitando ad un numero basso (tra 0 e un decimo del numero di tavoli) i livelli molto alti (sotto il tre) assegnati ai tavoli.
Questa istanza di test è stata ripetuta tre volte per 30 esecuzioni ciascuna.
I grafici sulle distribuzioni di giochi e livelli hanno confermato le premesse. Nella Tabella 6.1 vengono riassunti i risultati ottenuti.

Tabella 6.1: Risultati del primo test prestazionale

	Test 1	**Test 2**	**Test 3**
Presenza del dummy worker	4 / 30	8 / 30	7 / 30
Tempo di esecuzione medio	3.410s	3.662s	4.526s
Percentuale di lavoratori iniziali	+58.9%	+57.8%	+60.2%
Percentuale di lavoratori ottimizzati	+35.5%	+34.3%	+35.2%

L'analisi delle soluzioni non ammissibili e degli input che le hanno generate hanno evidenziato che in un caso non era stata trovata una soluzione ammissibile a causa di una cattiva rotazione delle limitate risorse disponibili per coprire alcune postazioni; risolvendo il problema a mano, con un'oculata rotazione era possibile creare una soluzione ammissibile, compiendo però delle scelte che vanno contro la scelta greedy. Si riconosce il limite dell'euristica.

Negli altri casi, le soluzioni non ammissibili sono dovute a un input iniziale non ben distribuito.

Secondo test prestazionale

Il test viene eseguito sulla seguente distribuzione dei livelli delle postazioni: i tavoli di livello alto (sotto il 3) sono limitati ad un numero basso (tra 0 e un decimo del numero dei tavoli); gli altri tavoli sono tutti di livello molto basso (8).
Questa istanza di test è stata ripetuta tre volte per 30 esecuzioni ciascuna.
I grafici sulle distribuzioni di giochi e livelli hanno confermato le premesse. Nella Tabella 6.2 vengono riassunti i risultati ottenuti.

Tabella 6.2: Risultati del secondo test prestazionale

	Test 1	**Test 2**	**Test 3**
Presenza del dummy worker	5 / 30	6 / 30	7 / 30
Tempo di esecuzione medio	4.197s	5.387s	4.205s
Percentuale di lavoratori iniziali	+68.2%	+68.3%	+66.1%
Percentuale di lavoratori ottimizzati	+35.6%	+34.5%	+36.3%

L'analisi delle soluzioni non ammissibili non ha rivelato criticità, se non per un caso di seguito descritto, in quanto sono dovute a un input iniziale non ben distribuito.
Tuttavia è stata osservata un'anomalia all'interno di una soluzione non ammissibile: due lavoratori che si alternano in modo anomalo su un tavolo rimanendo in pausa per moltissimi turni consecutivi. Ciò è dovuto al fatto che sono inspector di livello 1 e c'è solo un pit di livello 1. L'algoritmo preferisce mettere altri lavoratori meno qualificati nei pit di livello 2-3, perciò dopo che i due lavoratori in esame vanno in pausa vengono raramente assegnati ad un altro pit. Ciò preclude loro la possibilità di tornare a lavorare nell'unico pit di livello 1, perché dopo la pausa devono cambiare per forza postazione. A ciò è dovuto anche l'utilizzo del dummy worker. Si riconosce il limite dell'euristica.

Terzo test prestazionale

Il test viene eseguito sulla seguente distribuzione dei livelli delle postazioni: i tavoli di livello alto (sotto il 3) sono limitati ad un numero basso (tra 0 e un decimo del numero dei tavoli); gli altri tavoli sono tutti di livello molto basso (8). Inoltre tutti i pit sono di livello 3. Questa istanza di test è ritenuta la più importante perché rispetta al meglio la distribuzione dei tavoli all'interno del casinò di Londra.
Questa istanza di test è stata ripetuta tre volte per 30 esecuzioni ciascuna. I grafici sulle distribuzioni di giochi e livelli hanno confermato le premesse. Nella Tabella 6.3 vengono riassunti i risultati ottenuti.

Tabella 6.3: Risultati del terzo test prestazionale

	Test 1	**Test 2**	**Test 3**
Presenza del dummy worker	0 / 30	1 / 30	1 / 30
Tempo di esecuzione medio	6.043s	5.541s	5.307s
Percentuale di lavoratori iniziali	+71.0%	+68.7%	+68.7%
Percentuale di lavoratori ottimizzati	+36.9%	+37.9%	+37.4%

L'analisi delle soluzioni non ammissibili non ha rivelato criticità, in quanto sono dovute a un input iniziale non ben distribuito.

6.2.3 Conclusioni

L'analisi delle soluzioni (ammissibili e non) prodotte dall'algoritmo è stata, in conclusione, soddisfacente.
L'algoritmo rispetta sempre i vincoli imposti dal problema e crea delle soluzioni buone sotto i seguenti punti di vista:

1. dopo che un lavoratore è stato in pausa, l'algoritmo prioritizza il suo rientro nello scheduling, evitando di lasciarlo in pausa per più turni consecutivi;
2. i turni dei lavoratori sono solitamente di tre o quattro scaglioni consecutivi; trovare cinque scaglioni consecutivi è raro (con un numero di lavoratori congruo);
3. quando un lavoratore è schedulato per un turno in un certo tavolo, viene schedulato per quel tavolo anche nei turni immediatamente successivi per limitarne gli eccessivi spostamenti;
4. a pari disponibilità di due lavoratori per una postazione, viene sempre scelto il lavoratore con il livello più adatto a ricoprire la posizione.

Nonostante siano stati individuati dei limiti per l'algoritmo, essi sono stati ritenuti accettabili. Bisogna infatti ricordare che si tratta di un algoritmo euristico volto a trovare uno scheduling iniziale ammissibile che faccia da base ad altri algoritmi metaeuristici, quindi non solo per sua natura non può certamente trovare la soluzione ottima, ma anche gli è concesso di compiere scelte greedy che a posteriori si rivelano, a volte, poco oculate. Sarà compito di una futura metaeuristica sopperire alle mancanze del corrente algoritmo greedy.

Capitolo 7

Conclusioni

Questo capitolo di chiusura alla relazione illustra le conclusioni a cui si è giunti al termine dello stage. Vengono riportate sia conclusioni oggettive, come il raggiungimento degli obiettivi, il soddisfacimento dei requisiti, l'attuazione dell'analisi dei rischi, sia conclusioni soggettive circa l'esperienza di stage svolta.

7.1 Raggiungimento degli obiettivi

Come riportato nella Tabella 2.1 riguardante gli obiettivi da raggiungere a fine stage, da piano di lavoro lo stage prevedeva il soddisfacimento di dieci obiettivi obbligatori, due desiderabili e due opzionali. Al termine dello stage sono riuscita a raggiungere tutti gli obiettivi obbligatori, e il 50% degli obiettivi desiderabili e opzionali. In particolare, non ho soddisfatto, a causa della mancanza di tempo, gli obiettivi

* *De2 - Esplorazione di altri contesti applicativi*
* *Op2 - Implementazione con AMPL (o altro linguaggio di modellazione matematica) del modello di programmazione lineare intera per la soluzione esatta del problema di scheduling*

Per quanto riguarda il soddisfacimento dei requisiti riportati nelle tabelle contenute nel Capitolo 4, i risultati ottenuti sono riportati nella Tabella 7.1.

Tabella 7.1: Soddisfacimento dei requisiti

Tipo	**Individuati**	**Soddisfatti**
Requisiti totali	33	31
Requisiti obbligatori	27	27
Requisiti desiderabili	2	1
Requisiti opzionali	4	3

7.2 Resoconto dell'analisi dei rischi

Nella Tabella 7.2 viene riportata l'attuazione dell'analisi dei rischi.
L'analisi dei rischi completa può essere trovata nella Tabella 2.3.

Tabella 7.2: Attualizzazione dei rischi

Rischio	Verificato	Attualizzazione
Scelte di bad design nella progettazione del framework aziendale esistente, che possono rendere difficoltosa l'estensione.	Sì	Il template dell'algoritmo greedy si è rivelato in effetti poco estendibile, ciò ha reso necessaria una parziale riprogettazione sia del framework sia dell'estensione da me prodotta.
Bug all'interno del framework poiché alcune sue parti si trovano ancora in fase di sviluppo.	No	La parte del framework utilizzata per l'algoritmo greedy era già stata abbondantemente testata, perciò non ci sono stati problemi di malfunzionamenti.
Difficoltà nel confrontarsi con algoritmi euristici e problemi di ricerca operativa di cui si ha conoscenza nulla/scarsa	No	Le basi fornite dall'università di Ricerca Operativa e Algoritmi sono state più che sufficienti per non creare disorientamento studiando in maniera più approfondita gli argomenti dello stage. Non si sono dunque verificati ritardi sui tempi.

7.3 Consuntivo finale

Prima dell'inizio dello stage era stata svolta un'attività di pianificazione per stabilire quali sarebbero state le attività svolte e quanto sarebbe stato il tempo da dedicare loro (tale pianificazione è stata riportata all'interno della Tabella 2.2).

Per quanto riguarda i primi tre periodi, le ore dedicate ad ogni attività sono state congrue a quelle preventivate, a meno del risparmio di qualche ora; fanno eccezione le attività "*Ricerca ed analisi degli algoritmi più adatti*", "*Individuazione dei blocchi di base da algoritmi pre-esistenti*" e "*Integrazione dei blocchi*", infatti, come accennato nel Capitolo 3, in letteratura è difficile trovare un algoritmo che vada a creare uno scheduling iniziale per i problemi di schedulazione in maniera veloce.

Per quanto riguarda i periodi successivi, sussistono serie discrepanze per le attività di "*Implementazione*", "*Integrazione dell'algoritmo nelle librerie dell'azienda*" e "*Verifica e Testing*". Infatti, oltre alle problematiche sorte durante il periodo di verifica, già descritte all'interno del Capitolo 6 e che mi hanno fatto investire quasi il doppio del tempo previsto da piano di lavoro nella verifica e nei test, anche il tempo necessario per l'estensione del framework aziendale e implementazione dell'applicazione con l'algoritmo euristico si è rivelato maggiore del previsto, sia per una sottostima dell'impegno richiesto, sia soprattutto perché sono state incontrate delle difficoltà con l'integrazione dell'algoritmo greedy nel framework aziendale. Tali difficoltà, una volta esposte allo sviluppatore del framework, hanno indotto in una costruttiva discussione sulle scelte progettuali sia riguardanti il mio progetto che riguardanti il framework aziendale; tale discussione è risultata nella modifica della classe astratta `Greedy` da cui la classe `CasinoGreedy` ereditava, è stata quindi necessaria anche una sua parziale riprogettazione e reimplementazione. Le ore investite in più rispetto a quanto era stato pianificato per questa attività non sono state considerate una perdita da parte del tutor aziendale, in quanto è nell'interesse dell'azienda che il framework sia molto versatile nella risoluzione di problemi di Ricerca Operativa.

Infine, le ore dedicate a "*Implementazione del modello in AMPL*" e "*Confronto con i risultati dell'euristica e analisi delle presentazioni*" ammontano a zero, in quanto si è preferito dare priorità all'attività di verifica e testing, come concordato col tutor aziendale.

Il consuntivo finale completo, che confronta le ore preventivate e le ore effettivamente svolte, è riportato nella Tabella 7.3.

Tabella 7.3: Differenza ore consuntivo-preventivo

Attività	Preventivo	Consuntivo
1. Analisi del problema		
Assimilazione dei concetti di base per problemi di scheduling	10	7
Definizione caratteristiche del problema di scheduling da risolvere	10	6
2. Analisi dello stato dell'arte		
Studio della letteratura di base su problemi di scheduling	15	11
Ricerca ed analisi degli algoritmi più adatti	15	7
3. Ideazione e progettazione di un algoritmo euristico		
Individuazione dei blocchi di base da algoritmi pre-esistenti	15	7
Integrazione dei blocchi	10	3
Definizione dell'algoritmo di soluzione	15	11
4. Implementazione dell'algoritmo euristico		
Studio ed assimilazione del framework aziendale	15	17
Progettazione dei moduli	15	14
Implementazione	40	68
5. Integrazione e test dell'algoritmo		
Integrazione dell'algoritmo nelle librerie dell'azienda	30	49
Test preliminari su istanze semplificate	15	17
6. Confronto con tecniche esatte		
Studio dei modelli in letteratura	10	9
Definizione di un modello matematico	10	9
Implementazione del modello in AMPL	10	0
Confronto con i risultati dell'euristica e analisi delle presentazioni	10	0
7. Verifica e testing	20	37
8. Produzione di documenti, manuali e relazioni	35	28
	300	**300**

7.4 Valutazione personale e conoscenze acquisite

Dal punto di vista personale, svolgere e portare a termine questo progetto di stage è stato positivo. Ho trovato l'argomento dei problemi di scheduling che sono andata ad approfondire estremamente interessante, sia dal punto di vista teorico che dal punto di vista applicativo; inoltre una buona parte dell'implementazione del software che mi è stato richiesto di produrre era puramente algoritmica, che è l'aspetto dell'informatica che più mi appassiona e stimola. Essere quindi riuscita a produrre un software che rispettasse le specifiche, i vincoli e producesse un buono schedule è stato un risultato molto soddisfacente.
L'esperienza è stata positiva anche per quanto riguarda le conoscenze acquisite. Nonostante io non abbia dovuto apprendere da zero qualcosa di nuovo (se non l'utilizzo di Visual Studio, i fondamenti di Python per la scrittura dello script per l'esecuzione dei test e l'utilizzo di qualche libreria esterna), approfondire la mia conoscenza in materia di problemi di scheduling, modellazione e algoritmi greedy fino ad averne una padronanza tale da produrre il software è stato impegnativo e interessante.
Infine, valuto positivamente anche l'esperienza di inserimento all'interno dell'azienda, poiché lo stage si è svolto in un clima collaborativo e stimolante, in quanto tutti i membri del team si sono rivelati appassionati al progetto di cui fanno parte.
Ritengo quindi che svolgere lo stage presso Trans-Cel Autotrasporti sia stato per me molto formativo.

Glossario

Agile In ingegneria del software, è un insieme di metodi di sviluppo del software emersi a partire dai primi anni 2000 e fondati su insieme di principi comuni, direttamente o indirettamente derivati dai principi del *Manifesto per lo sviluppo agile del software* [15]. Tale manifesto si può riassumere in quattro punti:

1. le persone e le interazioni sono più importanti dei processi e degli strumenti;
2. è più importante avere software funzionante che documentazione;
3. bisogna collaborare con i clienti oltre che rispettare il contratto;
4. bisogna essere pronti a rispondere ai cambiamenti oltre che aderire alla pianificazione.

Un esempio di metodo di sviluppo di tipo agile è il metodo *scrum*[g]. 9

Algoritmo euristico I metodi di risoluzione dei problemi di ottimizzazione combinatoria ricadono in due categorie: i metodi esatti (vd. *Metodo esatto*) e metodi euristici. In matematica e informatica un metodo euristico è un particolare tipo di algoritmo progettato per risolvere un problema di ricerca operativa più velocemente (qualora i metodi esatti siano troppo lenti) o per trovare una soluzione approssimata (qualora i metodi esatti falliscano nel trovare una soluzione esatta). Il risultato viene ottenuto cercando di equilibrare ottimalità, completezza, accuratezza e velocità di esecuzione.
Possono essere applicati quando la soluzione è data selezionando il miglior sottoinsieme di un dato insieme di elementi e sono costruttivi, ovvero partono da una soluzione vuota e ad ogni iterazione aggiungono un elemento applicando dei criteri di selezione [14]. 3

Algoritmo Greedy Un algoritmo greedy è un algoritmo euristico che ad ogni iterazione compie una scelta "golosa", ovvero la migliore (in termini di impatto sulla funzione obiettivo) da prendere in quel momento. 10

AMPL È un linguaggio ad alto livello per descrivere e risolvere grossi e complicati problemi di programmazione matematica (per esempio problemi di ottimizzazione e di scheduling). 6

Business logic L'espressione logica di business (en. *business logic*) si riferisce a tutta la logica applicativa che rende operativa un'applicazione. Con tale nome ci si riferisce quindi all'algoritmica che gestisce lo scambio di informazioni tra una sorgente dati e l'interfaccia utente attraverso la logica di presentazione e le elaborazioni intermedie sui dati estratti. 47

Design pattern In informatica, per Design Pattern si intende un modello logico che rappresenta una soluzione progettuale generale ad un problema ricorrente. Durante le fasi di progettazione del software può essere utile applicare uno di questi modelli logici per risolvere un particolare problema che può presentarsi in più di una situazione. 36

Diagramma di Gantt Tipo di diagrammi molto usato per le attività di amministrazione di progetti poiché permettono una rappresentazione visuale di facile comprensione della durata delle attività che verranno svolte e le loro dipendenze. I Diagrammi di Gantt sono molto importanti per la gestione delle risorse di un progetto e per la creazione di un calendario delle attività che sia efficace. 6

Dummy worker Nel campo degli algoritmi euristici e dei problemi di scheduling, aggiungere un Dummy Worker è una tecnica usata per produrre sempre una soluzione ammissibile, anche quando in realtà l'algoritmo euristico non riesce a trovarne una. Il Dummy Worker può essere assegnato senza alcun vincolo a qualsiasi *Task* durante qualsiasi *TTB*, tuttavia inserirlo introduce una penalità altissima per la funzione obiettivo, di conseguenza viene introdotto in uno schedule quando non c'è nessun'altra alternativa. Nello schedule finale, il numero di *Task* non assegnati corrisponde al numero di *Task* assegnati al Dummy Worker. 20

Framework Nello sviluppo software, il framework è un'architettura logica di supporto su cui un software può essere progettato e realizzato; la sua funzione è quella di creare una infrastruttura generale, lasciando al programmatore il contenuto vero e proprio dell'applicazione. 3

Git Software di controllo versione distribuito utilizzabile da interfaccia a riga di comando. Rispetto a Git è meno flessibile, lievemente più lento ma con comandi più intuitivi e semplici da imparare. 9

Google Drive Servizio fornito da Google in ambiente cloud computing che offre funzioni di condivisione e hosting di file, permettendone la modifica collaborativa su uno spazio gratuito di 15 GB. 4

IDE È un software che, in fase di programmazione, aiuta il programmatore nello sviluppo del codice sorgente fornendo un controllo sintassi in tempo reale e una serie di strumenti e funzionalità di supporto allo sviluppo e al debugging. 11

Mercurial Software di controllo versione distribuito utilizzabile da interfaccia a riga di comando. È uno dei pù diffusi strumenti di controllo versione. 9

Meta-euristica In matematica e informatica è una strategia per esplorare in maniera efficiente lo spazio delle soluzioni per trovare delle soluzioni vicine all'ottimo e scappare dagli ottimi locali. Non sono algoritmi costruttivi e necessitano quindi di una soluzione per poterne trovarne un'altra migliore. 3

Metodo esatto I metodi di risoluzione dei problemi di ottimizzazione combinatoria ricadono in due categorie: i metodi esatti e metodi euristici (vd. *Algoritmo euristico*). I primi sono metodi che riescono a fornire una soluzione ottima, a differenza dei secondi che riescono a fornire una soluzione ammissibile ma senza la garanzia che sia ottimale [14]. Esempi di algoritmi esatti sono il simplesso o il branch-and-bound. 6

Milestone Traguardi intermedi ed importanti nello svolgimento di un progetto; sono spesso fissati in fase di pianificazione. 9

Modello incrementale In ingegneria del software, è una metodologia di sviluppo software in cui il cliente identifica, a grandi linee, i requisiti fondamentali e quelli desiderabili del prodotto software che vuole ottenere; viene poi deciso il numero di incrementi da effettuare, tenendo conto del fatto che ogni singolo incremento costituisce un sottoinsieme delle funzionalità del prodotto software. Una volta che gli incrementi sono stati identificati, si definiscono in dettaglio i requisiti che devono essere soddisfatti dagli incrementi prima di procedere allo sviluppo. 9

NP-Hard vd. *teoria della complessità computazionale*[g]. 1

Problema di Schedulazione/Scheduling Un problema di schedulazione definisce le variabili decisionali, una regione ammissibile entro la quale le variabili decisionali possono variare e una funzione obiettivo per determinare la migliore *schedulazione*[g] ammissibile. 1

Repository Ambiente di un sistema informativo in cui vengono gestiti i metadati attraverso tabelle relazionali. L'insieme di tabelle, regole e motori di calcolo tramite cui si gestiscono i metadati prende il nome di metabase. 9

Scatter Search La Scatter Search è una tecnica meta-euristica utilizzata per la soluzione di numerosi problemi di ottimizzazione, tra cui problemi di scheduling. Essa consiste nel mantenere un insieme di soluzioni candidate di buona qualità e molto diverse fra loro per poi ricombinarle linearmente per creare soluzioni migliori. 18

Schedulazione/Scheduling La schedulazione è una forma di processo decisionale che consiste nell'allocare risorse finite in modo tale che un dato obiettivo venga ottimizzato, sincronizzando e tempificando la sequenza delle operazioni. 1, 67

Scrum Metodo di sviluppo software che rientra fra i metodi agile. Prevede di dividere il progetto in blocchi rapidi di lavoro (Sprint) alla fine di ciascuno dei quali creare un incremento del software. Esso indica come definire i dettagli del lavoro da fare nell'immediato futuro e prevede vari meeting con caratteristiche precise per creare occasioni di ispezione e controllo del lavoro svolto. 9, 65

Supply chain La supply chain (in italiano, "catena di distribuzione") è il sistema di organizzazioni, persone, attività, informazioni e risorse coinvolte nel portare un prodotto o un servizio dal fornitore al cliente. 1

Tabu Search La Tabu Search è una tecnica meta-euristica utilizzata per la soluzione di numerosi problemi di ottimizzazione, tra cui problemi di scheduling. Essa consiste nel partire da una soluzione iniziale ed eseguire una serie di "mosse" che portano ad una nuova soluzione all'interno del vicinato per la quale la funzione obiettivo f assume un valore migliore del valore attuale. Per sfuggire gli ottimi locali, la Tabu Search permette un certo numero di mosse peggioranti rendendo tabù, proibite, le ultime mosse eseguite per evitare di "tornare indietro". 10

Teoria della complessità computazionale In informatica, la teoria della complessità computazionale è una branca della teoria della computabilità che studia le risorse minime necessarie (principalmente tempo di calcolo e memoria) per la risoluzione di un problema. I problemi sono classificati in differenti classi di complessità, in base all'efficienza del migliore algoritmo noto in grado di risolvere quello specifico problema:

* $\mathcal{P}$ (*Polynomial-time problems*) - Contiene tutti i problemi decisionali che possono essere risolti da una macchina di Turing deterministica in tempo polinomiale. La classe $\mathcal{P}$ può essere caratterizzata come quella classe di problemi per i quali si è in grado di trovare una soluzione in tempo polinomiale.
* $\mathcal{NP}$ (*Nondeterministic polynomial-time problems*) - Contiene tutti i problemi decisionali che possono essere risolti da una macchina di Turing non deterministica in tempo polinomiale. La classe $\mathcal{NP}$ può essere caratterizzata come quella classe di problemi per i quali non si è in grado di trovare una soluzione in tempo polinomiale, ma si è in grado di verificare se una possibile soluzione è effettivamente tale in tempo polinomiale.
  In particolare, $\mathcal{P} \subseteq \mathcal{NP}$.
* $\mathcal{NP} - \mathcal{C}$ o NP-Completi - Sono i più difficili problemi nella classe $\mathcal{NP}$: se si trovasse un algoritmo in grado di risolvere in tempo polinomiale un qualsiasi problema in $\mathcal{NP} - \mathcal{C}$, allora si potrebbe usarlo per risolvere in tempo polinomiale ogni problema in $\mathcal{NP}$.
  In particolare, $\mathcal{NP} - \mathcal{C} \subseteq \mathcal{NP}$.
* $\mathcal{H}$ o NP-Hard (*Nondeterministic polynomial-time hard problems*) - Classe di problemi (non limitata ai soli problemi di decisione, ma estesa anche ai problemi di ottimizzazione, ad esempio) che può essere definita informalmente come la classe dei problemi non meno difficili dei problemi $\mathcal{NP} - \mathcal{C}$, che a loro volta sono per definizione i più difficili della classe $\mathcal{NP}$. Si noti che non è sempre vero che $\mathcal{H} \subseteq \mathcal{NP}$.

[18] . 67

UML in ingegneria del software *UML, Unified Modeling Language* (ing. linguaggio di modellazione unificato) è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. L'*UML* svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. 2, 23

Variable Neighbourhood Search La Variable Neighbourhood Search è una tecnica meta-euristica utilizzata per la soluzione di numerosi problemi di ottimizzazione, tra cui problemi di scheduling. Essa consiste di due fasi: nella prima fase, trova una soluzione che costituisca un ottimo locale; nella seconda, perturba tale soluzione per cambiare "neighbourhood" e scappare dall'ottimo individuato. Da questa soluzione trova un ulteriore ottimo locale e lo confronta con quello precedentemente trovato; se esso è migliorante, la Variable Neighbourhood Search comincia a esplorare il suo neighbourhood, altrimenti continua ad esplorare il precedente. 19

XLS I file XLS sono fogli di calcolo creati con Microsoft Excel, la cui estensione è, per l'appunto, *.xls*. 40

Acronimi

AMPL A Mathematical Programming Language. 65

GUI Graphical User Interface. 9

IDE Integrated Developement Environment. 66

UML Unified Modeling Language. 68

Bibliografia

Riferimenti bibliografici

[1] Alessandro Agnetis. *Introduzione ai problemi di scheduling*. Dipartimento di Ingegneria dell'Informazione - Università di Siena, 2000 (cit. a p. 13).

[4] Broos Maenhout, Mario Vanhoucke. *A Hybrid Scatter Search Heuristic for Personalized Crew Rostering in the Airline Industry*. Faculteit Economie en Bedrijfskunde, 2007 (cit. alle pp. 13, 18).

[7] Edmund Burke, Timothy Curtois, Gerhard Post, Rong Qu, Bart Veltman. *A Hybrid Heuristic Ordering and Variable Neighbourhood Search for the Nurse Rostering Problem*. School of Computer Science e Information Technology - University of Nottingham, 2005 (cit. alle pp. 13, 18).

[8] Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1995 (cit. a p. 41).

[13] Ivo Blöchliger. *Modeling staff scheduling problems. A tutorial*. European Journal of Operational Research, 2003 (cit. a p. 13).

[14] Luigi De Giovanni. *Method and Models for Combinatorial Optimization*. Dipartimento di Matematica - Università degli studi di Padova (cit. alle pp. 65, 66).

[17] Tanguy Lapègue, Odile Bellenguez-Morineau, Damien Prot. *Method for the shift design and personnel task scheduling problem*. MOSIM 2014, 10ème Conférence Francophone de Modélisation, Optimisation et Simulation, 2014 (cit. alle pp. 13, 20).

Siti web consultati

[2] *Astah Professional*. URL: http://astah.net/editions/professional (cit. a p. 10).

[3] *Bitbucket*. URL: https://bitbucket.org/ (cit. a p. 9).

[5] *CodeProject*. URL: https://www.codeproject.com/Articles/42504/ExcelFormat-Library (cit. a p. 40).

[6] *Doxygen*. URL: http://www.stack.nl/~dimitri/doxygen/ (cit. a p. 10).

[9] *GitHub*. URL: https://github.com/ (cit. a p. 9).

[10] *GitLab*. URL: https://about.gitlab.com/ (cit. a p. 9).

[11] *Google Calendar*. URL: https://www.google.com/calendar (cit. a p. 10).

[12] *Google Drive*. URL: https://www.google.com/intl/it%5C_ALL/drive/ (cit. a p. 10).

[15] *Manifesto per lo sviluppo agile del software*. URL: http://agilemanifesto.org/iso/it/manifesto.html (cit. a p. 65).

[16] *Taiga*. URL: https://taiga.io/ (cit. a p. 9).

[18] *Teoria della complessità computazionale*. URL: https://it.wikipedia.org/wiki/Teoria_della_complessit%C3%A0_computazionale (cit. a p. 68).

[19] *Wolfram Alpha*. URL: https://www.wolframalpha.com/examples/StatisticalDistributions.html (cit. a p. 48).