Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Sviluppo di un modulo software per la gestione degli ordini di acquisto con l'utilizzo di metodi euristici di ottimizzazione

Tesi di laurea

Relatore
Prof. Luigi De Giovanni

Laureando
Filippo Brugnolaro
Matricola 1217321

Anno Accademico 2021-2022

La disumanità del computer sta nel fatto che,
una volta programmato e messo in funzione,
si comporta in maniera perfettamente onesta.

— *Isaac Asimov*

Ringraziamenti

In primis vorrei esprimere la mia gratitudine al Professor Luigi De Giovanni, relatore della mia tesi, per la disponibilità e l'aiuto fornitomi durante la stesura.

Desidero ringraziare con affetto la mia famiglia per tutto il sostegno e la vicinanza dimostrata in ogni momento e per non avermi mai fatto mancare nulla durante gli anni di studio.

Vorrei ringraziare i miei amici che mi sono stati vicini e mi hanno accompagnato in questi anni, soprattutto nei momenti difficili.

Infine desidero ringraziare in maniera speciale il mio amico Alessandro, che mi ha reso lo studio meno faticoso e con cui ho passato dei bei momenti, e Linpeng, che mi ha pazientemente guidato all'inizio del corso di laurea.

Padova, Settembre 2022

Filippo Brugnolaro

Indice

Elenco delle figure

Elenco delle tabelle

Lista degli algoritmi

Capitolo 1

Introduzione

Nel seguente capitolo si introduce brevemente l'azienda ospitante e il progetto affrontato.

1.1 L'azienda

Ergon Informatica S.R.L.[1](da qui in poi "*Ergon*") è un'azienda italiana, fondata nel 1988, con sede a Castelfranco Veneto.
Essa si occupa principalmente di soluzioni gestionali per piccole e medie imprese e dello sviluppo di *software* *ERP* per i settori dell'alimentare e dei trasporti, ma completa l'offerta con la vendita di prodotti hardware, servizi *web* e *hosting*, nonché con progetti di *server consolidation* e virtualizzazione di sistemi. L'azienda inoltre si è sviluppata in maniera costante negli anni e oggi può vantare una posizione di tutto rispetto tra le aziende dello stesso settore. Attualmente fanno parte della stessa gestione:

* *Ergon Informatica S.R.L.*: che si occupa del *software*;
* *Ergon S.R.L.*: che si occupa dei servizi tecnologici;
* *Ergon Servizi S.R.L.*: che si occupa dei servizi amministrativi, logistici e di *marketing* delle altre due parti.

Il logo dell'azienda è illustrato in Figura 1.1.

Figura 1.1: Logo Ergon Informatica S.R.L.

Il prodotto di proprietà dell'azienda è *ERGDIS*, sistema ERP il cui insieme dei moduli copre ogni aspetto della conduzione aziendale. Alcuni di essi, inoltre, si possono interfacciare con dispositivi automatici presenti in azienda, come, ad esempio, linee di confezionamento o *robot*.

[1]Sito ufficiale: https://www.ergon.it/

In particolare vengono gestiti compiti che si dislocano in vari ambiti e i moduli vengono dunque raggruppati nelle seguenti categorie:

- Amministrazione e Finanza
- Controllo di Gestione
- Area Acquisti
- Logistica
- Vendite
- Produzione
- *Web*
- *Business Intelligence*
- Qualità
- Gestione Archivi e Documentazione
- Pianificazione Consegne
- Area Mobile

In generale l'azienda può contare su una vasta gamma di clienti, in quanto i prodotti vengono sviluppati in base alle esigenze di ognuno di essi. Il prodotto viene prima creato a partire da uno *standard* a cui vengono successivamente aggiunte le varie funzionalità.
Il fatto di poter creare dei prodotti *custom*, rende l'azienda altamente competitiva e proprio per questo motivo il contatto continuo con gli *stakeholders* è molto importante sia per accontentare le loro richieste che per far evolvere *ERGDIS* in una maniera tale da essere sempre in linea con le esigenze di mercato.

1.2 L'idea

Lo *stage* proposto consiste nella progettazione e nello sviluppo di un modulo *software* volto ad assistere l'azienda nella fase di approvvigionamento dei prodotti dai propri fornitori, supportandola nel scegliere da quale fornitore e quando acquistare i prodotti. Questa nuova funzionalità andrebbe ad ottimizzare un modulo già esistente facente parte della gestione dell'*Area Acquisti*.
In pratica, per ogni prodotto da ordinare, viene presa in considerazione l'ultima data d'ordine disponibile prima dell'inizio dell'effettiva copertura del fabbisogno del prodotto stesso. Questo dunque non garantirebbe con certezza una scelta ottimale in relazione alle possibilità d'ordine fornite dagli appositi listini e calendario dei fornitori.

Data la natura combinatoria del problema, il modulo dovrà fornire in tempi ragionevoli una "buona soluzione" del problema, ovvero tendente il più possibile all'ottimo, e dovrà integrarsi con l'intero sistema *ERGDIS*.

È previsto inoltre che i dati su cui si è eseguita l'ottimizzazione e il confronto dei risultati vengano visualizzati tramite un'apposita interfaccia grafica che verrà sviluppata in linea con l'ambiente di sviluppo dell'azienda (*.NET Framework* e *DevExpress*).

1.3 Descrizione dello stage

1.3.1 Introduzione

Lo *stage* consiste nello sviluppo di un algoritmo di ottimizzazione che riesca a diminuire l'eventuale spesa che l'azienda andrebbe a sostenere.
L'algoritmo dovrà ottimizzare i risultati calcolati da un modulo preesistente, facente parte della gestione dell' *Area Acquisti*, il quale consiglia l'ordinamento della merce basandosi sull'ultima data d'ordine disponibile.

La scelta dell'attuale modulo infatti porterebbe queste conseguenze:

* **Inconsiderazione dell'andamento del mercato**: i prezzi sono variabili e dipendono dal periodo nel quale si comprano i prodotti.
* **Inconsiderazione dei bonus**: i fornitori possono garantire dei bonus in base al raggiungimento di determinati obiettivi di rapporti commerciali.

Facciamo un esempio per chiarire in maniera esplicita perchè si devono andare a considerare parametri come questi. Il problema è molto semplificato e verrà discusso nei capitoli successivi.

Articolo	Quantità	Data inizio copertura	Data fine copertura
FEG10	2	01/07/2022	07/07/2022
FRE02	3	06/07/2022	07/07/2022

Tabella 1.1: Esempio - Fabbisogni

Articolo	Fornitore	Data inizio validità	Data fine validità	Prezzo(€)
FEG10	47040	15/06/2022	21/06/2022	9.50
FRE02	47040	15/06/2022	21/06/2022	10.00
FEG10	46613	22/06/2022	31/12/9999	10.00
FRE02	46613	22/06/2022	31/12/9999	9.50

Tabella 1.2: Esempio - Listino Prezzi

Articolo	Fornitore	Data spedizione	Data di arrivo
FEG10	47040	17/06/2022	17/06/2022
FRE02	47040	19/06/2022	19/06/2022
FEG10	46613	31/06/2022	31/06/2022
FRE02	46613	03/07/2022	03/07/2022

Tabella 1.3: Esempio - Calendario spedizioni

Il modulo attuale avrebbe sicuramente ordinato entrambi gli articoli dal fornitore 46613, per un totale di 48.50€. Tuttavia è evidente che non è la soluzione ottima. Infatti sarrebbe stato opportuno ordinare l'articolo FEG10 dal fornitore 47040 e il FRE02 dal fornitore 46613, ottenendo dunque un totale di 47.50€.
Sebbene possa sembrare un risparmio minimo e trascurabile, se applicato a enormi quantità, può diventare un notevole risparmio di risorse.

Dopo queste considerazioni è chiaro come il modulo preesistente non garantisca necessariamente una soluzione ottima in termini economici ed è il motivo per il quale si è deciso di realizzare un nuovo modulo che vada a considerare determinati parametri.

Oltre a ciò, i risultati devono poter essere visualizzati tramite una WinForm in cui si dovrà far scegliere all'utente anche le date entro cui si vuole fare l'analisi.

Per l'azienda questo stage rappresenta un'opportunità per fornire un servizio aggiuntivo ai propri clienti, ma serve anche per avere una base dalla quale poter eventualmente estendere il modulo con nuovi algoritmi più efficienti.

1.3.2 Obiettivi

Di seguito vengono elencati tutti gli obiettivi previsti dallo *stage*:

* Analisi del contesto *ERGDIS*;
* Studio dei principali algoritmi di ricerca operativa e ottimizzazione combinatoria;
* Redazione di uno studio di fattibilità con integrazione di micro-moduli di test;
* Redazione di un'analisi dei requisiti;
* Sviluppo e codifica del modulo *software* con le tecnologie utilizzate dall'azienda;
* Redazione di documentazione tecnica riguardante le scelte implementative e architetturali effettuate;
* *Report* finale sui risultati ottenuti.

1.3.3 Analisi preventiva dei rischi

Durante la fase iniziale dello *stage* sono stati rilevati dei possibili rischi che avrebbero potuto presentarsi durante il percorso del progetto.
Si sono dunque trovate delle soluzioni che potessero arginare i problemi. In particolare:

1. **Comprensione e confronto degli algoritmi**

   Problema: il progetto richiede un'ampia fase di studio che riguarda principalmente la teoria di tecniche per la risoluzione di problemi di ricerca operativa e ottimizzazione combinatoria. Questo poteva portare a presentarsi la possibilità di non comprendere fino in fondo l'algoritmo e poteva essere difficile cogliere e confrontare i pregi e difetti di ciascuno di essi.

   Soluzione: è stato organizzato un incontro iniziale con il tutor per fornire una base da cui poi iniziare una ricerca più approfondita. Sono state fornite anche delle dispense utili per rafforzare la base di partenza.

2. **Tecnologie e ambiente di sviluppo**

   Problema: venivano richieste alcune tecnologie, come per esempio *Entity Framework* o *DevExpress* a me assolutamente ignote. Sebbene avessi delle basi abbastanza solide di *C#* derivanti dalla conoscenza di altri linguaggi quali *C++* e *Java*, venivano richieste tuttavia alcune tecnologie integrate nel linguaggio (*LINQ*), anch'esse ignote. L'ambiente di sviluppo e l'*IDE* non erano mai stati utilizzati.

   Soluzione: sono stati forniti dei riferimenti consigliati per l'autoapprendimento. Tuttavia qualsiasi dubbio ragionevolmente particolare poteva essere richiesto al tutor. È stato effettuato insieme al tutor il *setup* dell'ambiente di sviluppo e la conseguente creazione dei *database*.

3. **Calibrazione dei parametri e funzione obiettivo**

   Problema: dopo la scelta e l'implementazione dell'algoritmo, è molto importante:

   * definire una funzione obiettivo che vada a descrivere in maniera "buona" l'andamento dell'algoritmo stesso;
   * calibrare i parametri in base allo spazio delle soluzioni del problema preso in esame.

Entrambe sono azioni molto delicate che possono compromettere il funzionamento stesso dell'algoritmo anche se implementato correttamente.

Soluzione: cercare una costruzione e calibrazione per passi e presentarle in una discussione con il tutor, in modo tale da creare una *baseline* su cui basarsi per continuare con i passi successivi.

1.4 Organizzazione del testo

Di seguito viene illustrata l'organizzazione dei capitoli successivi:

Il secondo capitolo approfondice lo studio di fattibilità effettuato, utile per entrare a conoscenza delle più utilizzate tecniche di ottimizzazione combinatoria e per analizzare quali siano i vantaggi e svantaggi di ognuno di essi.

Il terzo capitolo descrive l'analisi dei requisiti del progetto, comprensiva di diagrammi dei casi d'uso e raccolta dei requisiti derivanti dall'analisi di questi ultimi.

Il quarto capitolo approfondisce le fasi di progettazione e codifica, comprensiva di diagrammi delle classi e di approfondimenti a livello implementativo.

Il quinto capitolo espone tutte le verifiche effettuate durante il progetto e la validazione finale a conferma dei requisiti inizialmente stilati nella fase di analisi dei requisiti.

Il sesto capitolo presenta le conclusioni tratte dallo *stage*, comprensivo di conoscenze acquisite e considerazioni di carattere personale.

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 alla fine del presente documento;
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Studio di fattibilità

In questo capitolo viene esposto lo studio di fattibilità, in cui verranno evidenziati i punti critici, i vantaggi e gli svantaggi delle soluzioni analizzate

2.1 Introduzione allo studio

Lo studio di fattibilità rappresenta una delle parti più corpose del progetto in quanto viene richiesto un ampio periodo di autoapprendimento dei i principali algoritmi di ricerca operativa e di ottimizzazione combinatoria, seguito da un'ulteriore approfondimento attraverso la consultazione di vari *paper*[2] disponibili online.

Questa prima parte definirà una prima scelta tra gli algoritmi disponibili in base alle informazioni reperite.

La seconda parte invece consiste nello sviluppo di micro-moduli di test sulle scelte effettuate precedentemente in modo tale da poter effettuare una analisi ed un confronto accurati basati su parametri che verranno definiti successivamente.

Considerando anche il rapporto tra costi e risorse, le soluzioni che sono state identificate come le più plausibili e sottoposte a uno studio più approfondito sono:

1. Algoritmo greedy
2. Tabu search
3. Algoritmo genetico

Prima di proseguire con lo studio di fattibilità, è necessario dichiarare le metriche che sono state utilizzate per effetuare un confronto equo tra gli algoritmi. Chiaramente esse derivano dagli obiettivi che l'algoritmo deve soddisfare per risolvere il problema.
Vengono elencati in seguito i parametri:

* **Efficienza**: la capacità dell'algoritmo di utilizzare meno risorse possibili per risolvere il problema.
* **Efficacia**: la capacità dell'algoritmo di risolvere il problema fornendo una soluzione il più possibile corretta.
* **Complessità implementativa**: quantitavo di risorse temporali impiegate per sviluppare l'algoritmo.
* **Paper**: dichiarazioni o dati di esperimenti già effettuati.

[2]Verranno citati nella bibliografia solo quelli ritenuti più significativi

2.2 Soluzioni proposte

Per ogni soluzione proposta viene effettuata una breve introduzione, seguita da vantaggi e svantaggi. Gli pseudocodici che descrivono in maniera sintetica i micro-moduli di test si basano su un problema di *string replacement* con input di lunghezza uguale.
Si è volutamente scelto questo tipo di problema molto semplice poichè, se i micro-moduli fossero stati sviluppati intercalandoli all'interno del contesto, si avrebbe avuto un enorme spreco di risorse temporali.

2.2.1 Algoritmo Greedy

L'algoritmo *greedy* ("*goloso*") viene così chiamato poichè basa la ricerca di una soluzione ammissibile ottima sulla scelta, secondo un criterio predefinto, della miglior soluzione disponibile ad ogni passo senza rimettere in discussione la scelta appena effettuata.
Di seguito viene presentato lo pseudocodice dell'algoritmo greedy.

Pseudocodice string replacement - Algoritmo Greedy

Input: $stringa\_corretta$, $stringa\_errata$
Output: $funzione\_obiettivo$

```
1:  procedure My_Greedy_Algorithm(stringa_corretta, stringa_errata)
2:      array_str_err ← generate_array(stringa_errata)
3:      array_str_corr ← generate_array(stringa_corretta)
4:      funzione_obiettivo ← array_str_corr.Length
5:      pos ← 0
6:      for each element ∈ array_str_err do
7:          if element ≠ array_str_corr[pos] then
8:              element = array_str_corr[pos]
9:          end if
10:         funzione_obiettivo ← funzione_obiettivo − 1
11:         pos ← pos + 1
12:     end for
13:     return funzione_obiettivo
14: end procedure
```

Osservazioni
Come si può notare, l'algoritmo greedy è molto intuitivo e, in questo caso, anche banale. Infatti, ad ogni iterazione, definiti x come l'elemento in posizione pos nella stringa errata e y come l'elemento, nella stessa posizione, nella stringa corretta, se $x \neq y$ si effettua un replace di x con la scelta migliore disponibile in quel momento, ovvero y stesso.

Aspetti positivi

* Bassa complessità implementativa
* Bassa complessità computazionale dell'algoritmo
* Possibilità di effettuare scelte greedy differenti in problemi di vaste dimensioni

Aspetti negativi

* Possibilmente inefficace, può non raggiungere l'ottimo globale a causa delle scelte greedy effettuate ad ogni iterazione che possono scartare soluzioni migliori nel lungo periodo

2.2.2 Tabu Search

La Tabu Search[3] è un metodo, classificato come *Trajectory Method*, basato su ricerca locale in grado di eludere l'intrappolamento del metodo in un minimo locale sfruttando costantemente la memoria.
In particolare, oltre a ricordare la migliore soluzione corrente, vengono salvate, in quella che viene definita *Tabu List*, anche le k mosse precedentemente effettuate in modo tale da non incombere nel rischio di un ciclo nel breve periodo.
Viene dunque orientata la ricerca tramite la modifica del vicinato in funzione della storia dell'esplorazione, ma anche tramite la diversificazione dei sottospazi di ricerca tramite il passaggio per soluzioni non ammissibili.
Di seguito viene presentato lo pseudocodice della tabu search.

Pseudocodice string replacement - Tabu Search

Input: *stringa_corretta*, *stringa_errata*, *capienza_tabu_list*, *max_iterazioni*
Output: $funzione\_obiettivo$

```
1: procedure MY_TABU_SEARCH(stringa_corretta, stringa_errata, capienza_tabu_list,
   max_iterazioni)
2:     sol_curr ← stringa_errata
3:     funzione_obiettivo_curr ← stringa_corretta.Length
4:     funzione_obiettivo_best ← funzione_obiettivo_curr
5:     conta ← 0
6:     while conta < max_iterazioni do
7:         vicinato ← genera_vicinato()
8:         while vicinato.Length > 0 and conta < max_iterazioni do
9:             vicino_migl ← ottieni_miglior_vicino(vicinato)
10:            if vicino_migl ∉ tabu_list then
11:                funzione_obiettivo_curr ← calculate_fo(vicino_migl)
12:                if funzione_obiettivo_curr < funzione_obiettivo_best then
13:                    Inserisci vicino_migl nella tabu_list considerando la capienza_tabu_list
14:                    sol_curr ← vicino_migl
15:                    funzione_obiettivo_best ← funzione_obiettivo_curr
16:                    conta ← conta + 1
17:                    break
18:                end if
19:            end if
20:            Rimuovi vicino_migl dal vicinato
21:            conta ← conta + 1
22:        end while
23:    end while
24:    return funzione_obiettivo
25: end procedure
```

Osservazioni
Notiamo come il ruolo della *tabu_list* sia fondamentale in quanto, in caso di appartenza della mossa alla *tabu_list*, non fa calcolare e conseguentemente fare il confronto con $funzione\_obiettivo_{best}$. Inoltre se la $funzione\_obiettivo_{curr}$ non è migliorante, l'iterazione viene comunque contata.
Per quanto riguarda la condizione di **break**, essa viene messa poichè, avendo trovato una soluzione migliorante, non è più necessario esplorare il vicinato corrente ed è dunque necessario crearne uno nuovo a partire dalla nuova soluzione.

[3]Per una descrizione più accurata si legga la sezione x

Aspetti positivi

* Bassa complessità implementativa
* Bassa complessità computazionale dell'algoritmo
* Velocità di raggiungimento del minimo locale
* Fuga da ottimi locali

Aspetti negativi

* Possibilmente inefficace, può non raggiungere l'ottimo globale
* Difficile calibrazione dei parametri
* Numero di iterazioni necessario può essere molto alto
* Necessità di una soluzione iniziale

2.2.3 Algoritmo Genetico

L'algoritmo genetico è un metodo, classifcato come *population based*, basato sul concetto che la natura abbia la tendenza ad organizzarsi in strutture ottimizzate, in gran parte ispirato alle teorie sull'evoluzione di Charles Darwin[4].
In particolare, ad ogni iterazione, non viene mantenuta una sola soluzione, ma un'insieme di soluzioni, definita anche come popolazione. Gli individui (soluzioni) vengono codificati tramite un cromosoma contenente una serie di geni (variabili decisionali del problema) e, per ogni individuo appartente alla popolazione, viene associata quella che viene definita come la sua idoneità, tramite l'utilizzo di una funzione di fitness, che guida il processo di selezione, basato su metodi probabilistici (es: *metodo Montecarlo*, *linear ranking*, *torneo-n*).
Prima di ogni iterazione vengono dunque presi gli individui e verranno accoppiati tramite degli operatori di ricombinazione (es: *crossover uniforme*, *cut-point crossover*, *mutazione...*) per generare dei figli che assumeranno le migliori caratteristiche dei genitori. Alla fine dell'algoritmo verrà scelta la soluzione con la maggior fitness possibile.

Pseudocodice string replacement - Algoritmo Genetico

Input: $stringa\_corretta$, $crossover\_rate$, $mutation\_rate$, $max\_iterazioni$
Output: sol_{best}, $fitness_{best}$

1: **procedure** My_Genetic_Algorithm($stringa\_corretta$, $stringa\_errata$, $crossover\_rate$, $mutation\_rate$, $max\_iterazioni$)
2: $arr_{str\_corr} \leftarrow codifica(stringa\_corretta)$
3: $lista\_pop \leftarrow inizializza\_pop()$
4: $set\_fitness()$
5: $fitness_{best} \leftarrow most\_fitness\_val(arr_{str\_corr})$
6: $sol_{best} \leftarrow most\_fitness\_sol(arr_{str\_corr})$
7: $num_{crossover} \leftarrow calculate\_num\_crossover(crossover\_rate, lista\_pop)$
8: $conta \leftarrow 0$

[4]Scienzato conosciuto per le sue teorie sull'evoluzione secondo le quali gli individui di una popolazione sono in competizione fra loro e, in questa lotta per la sopravvivenza, l'ambiente opera una selezione naturale tramite la quale vengono eliminati gli individui più deboli, cioè quelli meno adatti a sopravvivere a determinate condizioni. Solo i più adatti sopravvivono e trasmettono i loro caratteri ai figli.

```
    while conta < max_iterazioni do
        lista_pop ← seleziona_individui(num_crossover)
        set_fitness()
        i ← 0
        while i < num_crossover do
            lista_genitori ← seleziona_individui(2)
            figlio ← genera_figlio(lista_genitori)
            Aggiungi il figlio in coda alla lista nuova_pop
            i ← i + 1
        end while
        lista_pop ← nuova_pop
        lista_pop ← mutazione_rand(lista_pop)
        set_fitness()
        fitness_curr ← most_fitness_val(arr_str_corr)
        sol_curr ← most_fitness_sol(arr_str_corr)
        if fitness_curr > fitness_best then
            fitness_best ← fitness_curr
            sol_best ← sol_curr
        end if
        conta ← conta + 1
    end while
    return sol_best, fitness_best
end procedure
```

Osservazioni
Si noti come, per ogni popolazione, vengano scelti un numero di individui dipendente dal $crossover\_rate$ e vengano effettuati $num_{crossover}$ crossover scegliendo 2 individui dalla popolazione che fungono da genitori. Viene dunque creata una nuova popolazione di figli a cui viene anche applicata una mutazione a un figlio random per variare il patrimonio genetico, in modo tale da avere più possibilità di trovare buone caratteristiche.

Aspetti positivi

* Fuga da ottimi locali
* Analisi di più sottospazi delle soluzioni, grazie alla varietà della popolazione
* Utilizzo modelli probabilistici

Aspetti negativi

* Modesta complessità implementativa
* Complessità computazionale dell'algoritmo
* Possibilmente inefficace, può non raggiungere l'ottimo globale
* Difficile calibrazione dei parametri
* Difficile codifica degli individui in alcuni problemi
* Numero di iterazioni necessario può essere molto alto

2.3 Conclusioni dello studio

Lo studio è servito per capire come funzionassero gli algoritmi e quali fossero i loro pregi e difetti. Di seguito si hanno i risultati dell'analisi dei 3 algoritmi eseguiti singolarmente.

Si è optato per l'utilizzo della tabu search in quanto rappresentava un buon compromesso sia a livello di efficacia ed efficienza che a livello implementativo. Per quanto riguarda la soluzione iniziale, questa, data la sua bassa complessità computazionale, può essere generata a partire da un algoritmo greedy, in modo tale da avere già a disposizione una buona soluzione di base.
Sebbene l'algoritmo genetico fosse interessante per quanto ne concerne il mantenimento di più soluzioni ad ogni iterazione, in realtà l'algoritmo si è dimostrato molto più lento rispetto ai due precedenti. Inoltre un'alta complessità realizzativa lo ha portato dunque all'esclusione.
Tuttavia, sebbene la tabu search, ad ogni iterazione, possieda una sola soluzione, è stato pensato che l'algoritmo possa essere lanciato in più thread, riuscendo dunque a simulare il mantenimento di più soluzioni dell'algoritmo genetico, passando ai thread chiaramente soluzioni iniziali differenti.

Capitolo 3

Analisi dei requisiti

Breve introduzione al capitolo

3.1 Casi d'uso

Per lo studio dei casi di utilizzo del prodotto sono stati creati dei diagrammi. I diagrammi dei casi d'uso (in inglese *Use Case Diagram*) sono diagrammi di tipo UML dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso. Essendo il progetto finalizzato alla creazione di un tool per l'automazione di un processo, le interazioni da parte dell'utilizzatore devono essere ovviamente ridotte allo stretto necessario. Per questo motivo i diagrammi dei casi d'uso risultano semplici e in numero ridotto.

A livello formale, i diagrammi dei casi d'uso avranno la seguente forma:

UC<CodicePadre>.<CodiceFiglio>

È importante ribadire come questo formalismo sia gerarchico, ovvero un codice figlio può essere codice padre di un suo eventuale codice figlio. Possono essere figli le generalizzazioni e i sottocasi d'uso.

Nella figura di seguito verrà illustrato il diagramma del sistema principale con tutti i casi d'uso.

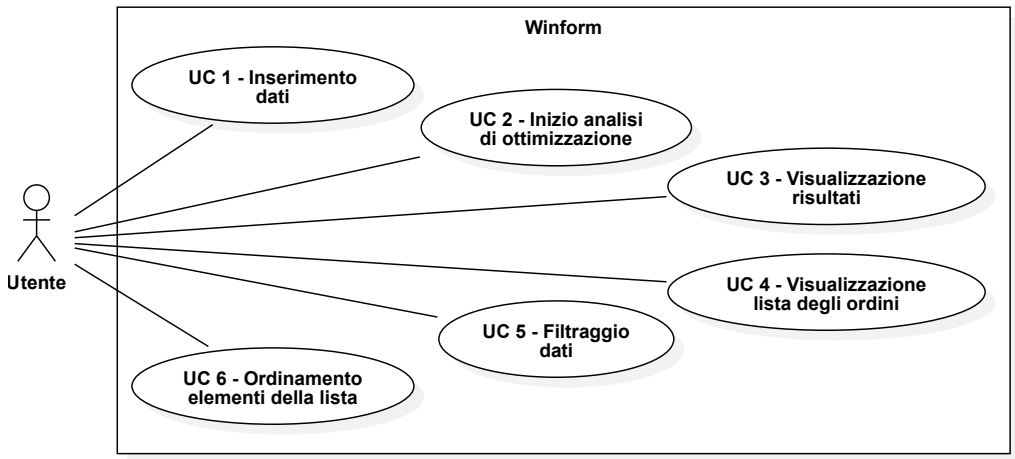

Figura 3.1: Use case - sistema principale

UC 1 - Inserimento dati

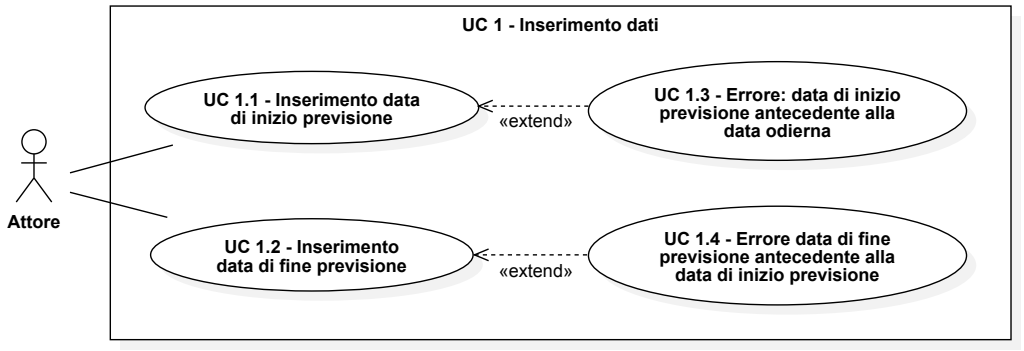

Figura 3.2: UC1 - Inserimento dati

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente è dentro la *form* e non ha ancora inserito alcun dato.
* **Scenario principale:**
  1. L'utente inserisce i dati.
* **Postcondizione:**

  L'utente ha inserito i dati correttamente.

UC 1.1 - Inserimento data di inizio previsione

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente è dentro la *form* e non ha ancora inserito la data di inizio previsione.
* **Scenario principale:**
  1. L'utente seleziona la data di inizio previsione.
* **Postcondizione:**

  L'utente ha inserito la data di inizio previsione correttamente.
* **Scenario alternativo:**
  - La *form* segnala un errore di immissione dati (UC 1.3).

UC 1.2 - Inserimento data di fine previsione

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente è dentro la *form* e non ha ancora inserito la data di fine previsione.
* **Scenario principale:**
  1. L'utente inserisce la data di fine previsione;
* **Postcondizione:**

  L'utente ha inserito la data di fine previsione correttamente.
* **Scenario alternativo:**
  - La *form* segnala un errore di immissione dati (UC 1.4).

UC 1.3 - Errore: data di inizio previsione antecedente alla data odierna

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente è dentro la *form* e ha inserito una data di inizio previsione antecedente alla data odierna.
* **Scenario principale:**
  1. L'utente conferma la data di inizio previsione;
  2. L'utente visualizza un errore generato dalla *form*.
* **Postcondizione:**

  L'utente viene avvisato dell'errore di immissione.

UC 1.4 - Errore: data di fine previsione antecedente alla data di inizio previsione

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente è dentro la *form* e ha inserito una data di fine previsione antecedente alla data odierna.
* **Scenario principale:**
  1. L'utente conferma la data di fine previsione;
  2. L'utente visualizza un errore generato dalla *form*.
* **Postcondizione:**

  L'utente viene avvisato dell'errore di immissione.

UC 2 - Inizio analisi di ottimizzazione

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente è dentro la *form* e ha inserito una data di inizio e fine previsione valide.
* **Scenario principale:**
  1. L'utente conferma l'inizio dell'analisi di ottimizzazione.
* **Postcondizione:**

  L'utente ha effettuato l'analisi di ottimizzazione per le date di inizio e fine previsione e visualizza correttamente i risultati (UC 3).

UC 3 - Visualizzazione risultati

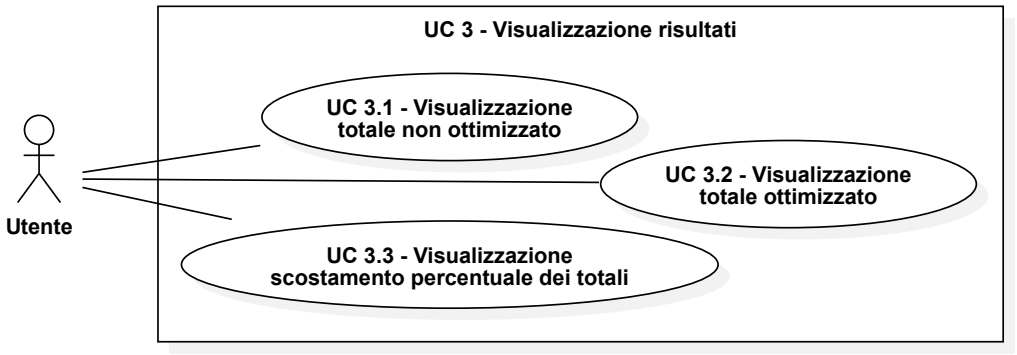

Figura 3.3: UC3 - Visualizzazione risultati

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente è dentro la *form* e ha effettuato l'analisi di ottimizzazione correttamente.
* **Scenario principale:**
  1. L'utente visualizza il totale non ottimizzato (UC 3.1);
  2. L'utente visualizza il totale ottimizzato (UC 3.2);
  3. L'utente visualizza lo scostamento percentuale dei totali (UC 3.3).
* **Postcondizione:**

  L'utente visualizza correttamente tutti i risultati.

UC 3.1 - Visualizzazione totale non ottimizzato

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente è dentro la *form* e ha effettuato l'analisi di ottimizzazione correttamente.
* **Scenario principale:**
  1. L'utente visualizza il totale non ottimizzato.
* **Postcondizione:**

  L'utente visualizza correttamente il totale non ottimizzato.

UC 3.2 - Visualizzazione totale ottimizzato

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente è dentro la *form* e ha effettuato l'analisi di ottimizzazione correttamente.
* **Scenario principale:**
  1. L'utente visualizza il totale ottimizzato.
* **Postcondizione:**

  L'utente visualizza correttamente il totale ottimizzato.

UC 3.3 - Visualizzazione scostamento percentuale dei totali

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente è dentro la *form* e ha effettuato l'analisi di ottimizzazione correttamente.
* **Scenario principale:**
  1. L'utente visualizza lo scostamento percentuale dei totali.
* **Postcondizione:**

  L'utente visualizza correttamente lo scostamento percentuale dei totali.

UC 4 - Visualizzazione lista degli ordini

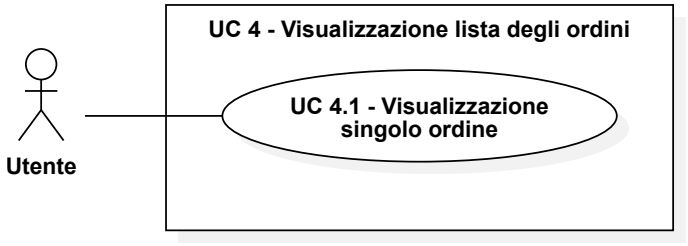

Figura 3.4: UC4 - Visualizzazione lista degli ordini

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente è dentro la *form* e ha effettuato l'analisi di ottimizzazione correttamente.
* **Scenario principale:**
  1. L'utente visualizza la lista degli ordini da effettuare.
* **Postcondizione:**

  L'utente visualizza correttamente la lista degli ordini da effettuare.

UC 4.1 - Visualizzazione singolo ordine

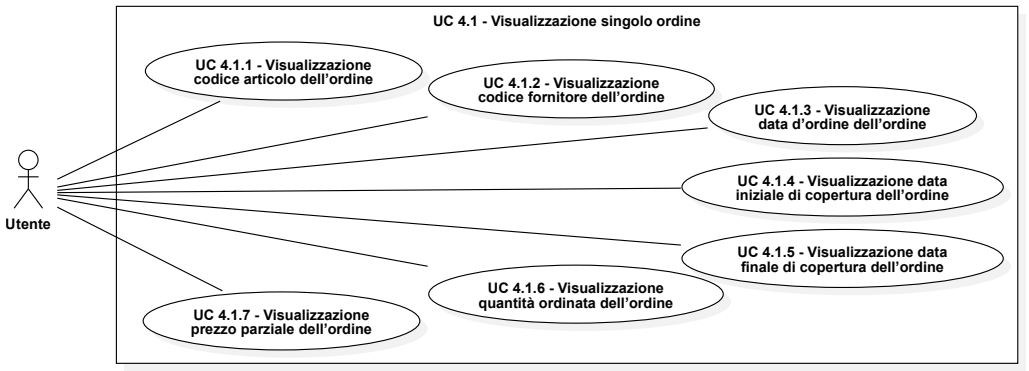

Figura 3.5: UC4 - Visualizzazione lista degli ordini

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente la lista degli ordini.

* **Scenario principale:**

  1. L'utente visualizza il singolo ordine con tutte le informazioni tra cui:
     - codice articolo (UC 4.1.1);
     - codice fornitore (UC 4.1.2);
     - data d'ordine (UC 4.1.3);
     - data iniziale di copertura (UC 4.1.4);
     - data finale di copertura (UC 4.1.5);
     - quantità ordinata (UC 4.1.6);
     - prezzo parziale (UC 4.1.7);

     .

* **Postcondizione:**

  L'utente visualizza correttamente il singolo ordine.

UC 4.1.1 - Visualizzazione codice articolo dell'ordine

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente il singolo ordine.
* **Scenario principale:**

  1. L'utente visualizza il codice articolo del singolo ordine.
* **Postcondizione:**

  L'utente visualizza correttamente il codice articolo del singolo ordine.

UC 4.1.2 - Visualizzazione codice fornitore dell'ordine

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente il singolo ordine.
* **Scenario principale:**

  1. L'utente visualizza il codice fornitore del singolo ordine.
* **Postcondizione:**

  L'utente visualizza correttamente il codice fornitore del singolo ordine.

UC 4.1.3 - Visualizzazione data d'ordine dell'ordine

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente il singolo ordine.
* **Scenario principale:**
  1. L'utente visualizza la data d'ordine del singolo ordine.
* **Postcondizione:**

  L'utente visualizza correttamente la data d'ordine del singolo ordine.

UC 4.1.4 - Visualizzazione data iniziale di copertura dell'ordine

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente il singolo ordine.
* **Scenario principale:**
  1. L'utente visualizza la data iniziale di copertura del singolo ordine.
* **Postcondizione:**

  L'utente visualizza correttamente la data iniziale di copertura del singolo ordine.

UC 4.1.5 - Visualizzazione data finale di copertura dell'ordine

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente il singolo ordine.
* **Scenario principale:**
  1. L'utente visualizza la data finale di copertura del singolo ordine.
* **Postcondizione:**

  L'utente visualizza correttamente la data finale di copertura del singolo ordine.

UC 4.1.6 - Visualizzazione quantità ordinata dell'ordine

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente il singolo ordine.
* **Scenario principale:**
  1. L'utente visualizza la quantità ordinata del singolo ordine.
* **Postcondizione:**

  L'utente visualizza correttamente la quantità ordinata del singolo ordine.

UC 4.1.7 - Visualizzazione prezzo parziale dell'ordine

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente il singolo ordine.
* **Scenario principale:**
  1. L'utente visualizza il prezzo parziale del singolo ordine.
* **Postcondizione:**

  L'utente visualizza correttamente il prezzo parziale del singolo ordine.

UC 5 - Filtraggio dati

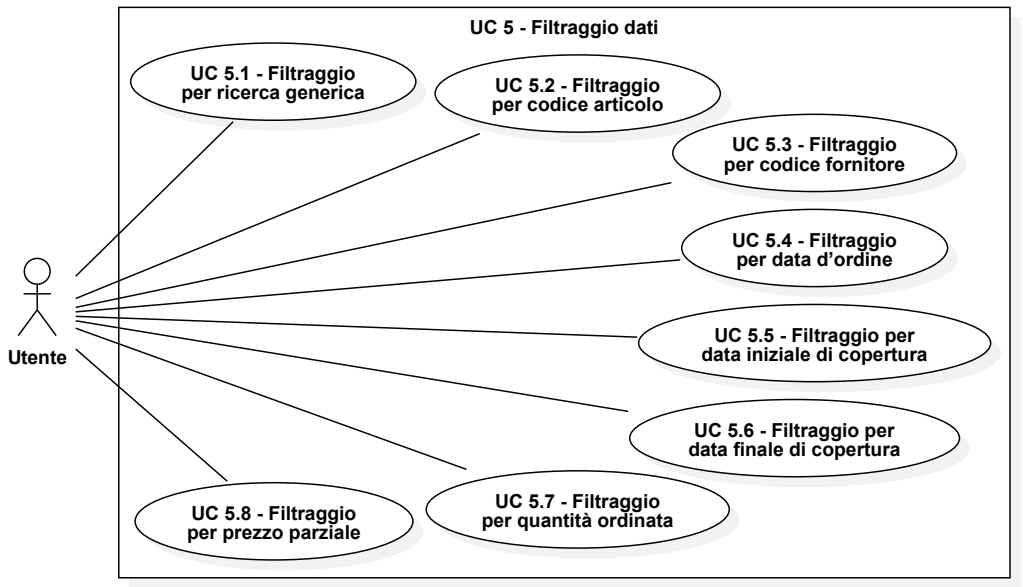

Figura 3.6: UC4 - Visualizzazione lista degli ordini

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente la lista degli ordini.
* **Scenario principale:**
  1. L'utente sceglie uno o più filtri da applicare alla lista.
     In particolare le tipologie di filtro disponibili sono per:
     - codice articolo (UC 5.1);
     - codice articolo (UC 5.2);
     - codice fornitore (UC 5.3);
     - data d'ordine (UC 5.4);
     - data iniziale di copertura (UC 5.5);
     - data finale di copertura (UC 5.6);
     - quantità ordinata (UC 5.7);
     - prezzo parziale (UC 5.8);
* **Postcondizione:**

  L'utente visualizza correttamente tutti gli elementi che soddisfano i filtri applicati.

UC 5.1 - Filtraggio per ricerca generica

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente la lista degli ordini.
* **Scenario principale:**
  1. L'utente filtra uno o più ordini tramite una ricerca generica.
* **Postcondizione:**

  L'utente visualizza correttamente tutti gli elementi che soddisfano il filtro.

UC 5.2 - Filtraggio per codice articolo

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente la lista degli ordini.
* **Scenario principale:**
  1. L'utente filtra uno o più ordini per codice articolo.
* **Postcondizione:**

  L'utente visualizza correttamente tutti gli elementi che soddisfano il filtro.

UC 5.3 - Filtraggio per codice fornitore

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente la lista degli ordini.
* **Scenario principale:**
  1. L'utente filtra uno o più ordini per codice fornitore.
* **Postcondizione:**

  L'utente visualizza correttamente tutti gli elementi che soddisfano il filtro.

UC 5.4 - Filtraggio per data d'ordine

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente la lista degli ordini.
* **Scenario principale:**
  1. L'utente filtra uno o più ordini per data d'ordine.
* **Postcondizione:**

  L'utente visualizza correttamente tutti gli elementi che soddisfano il filtro.

UC 5.5 - Filtraggio per data iniziale di copertura

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente la lista degli ordini.
* **Scenario principale:**
  1. L'utente filtra uno o più ordini per data iniziale di copertura.
* **Postcondizione:**

  L'utente visualizza correttamente tutti gli elementi che soddisfano il filtro.

UC 5.6 - Filtraggio per data finale di copertura

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente la lista degli ordini.
* **Scenario principale:**
  1. L'utente filtra uno o più ordini per data finale di copertura.
* **Postcondizione:**

  L'utente visualizza correttamente tutti gli elementi che soddisfano il filtro.

UC 5.7 - Filtraggio per quantità ordinata

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente la lista degli ordini.
* **Scenario principale:**
  1. L'utente filtra uno o più ordini per quantità ordinata.
* **Postcondizione:**

  L'utente visualizza correttamente tutti gli elementi che soddisfano il filtro.

UC 5.8 - Filtraggio per prezzo parziale

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente la lista degli ordini.
* **Scenario principale:**
  1. L'utente filtra uno o più ordini per prezzo parziale.
* **Postcondizione:**

  L'utente visualizza correttamente tutti gli elementi che soddisfano il filtro.

UC 6 - Ordinamento della lista degli ordini

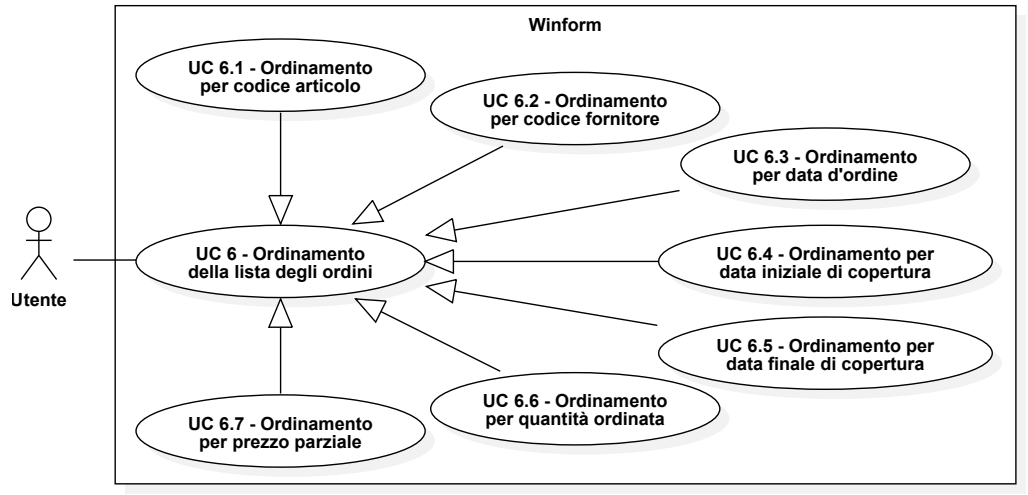

Figura 3.7: UC4 - Visualizzazione lista degli ordini

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente la lista degli ordini.
* **Scenario principale:**
  1. L'utente sceglie l'ordinamento da applicare alla lista.
* **Postcondizione:**

  L'utente visualizza correttamente tutti gli elementi ordinati secondo la sua scelta.
* **Generalizzazioni:**
  - Ordinamento per codice articolo (UC 6.1);
  - Ordinamento per codice fornitore (UC 6.2);
  - Ordinamento per data d'ordine (UC 6.3);
  - Ordinamento per data previsione inizio copertura (UC 6.4);
  - Ordinamento per data previsione fine copertura (UC 6.5);
  - Ordinamento per quantità ordinata (UC 6.6);
  - Ordinamento per prezzo parziale (UC 6.7).

UC 6.1 - Ordinamento per codice articolo

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente la lista degli ordini.

* **Scenario principale:**

  1. L'utente ordina la lista per codice articolo.

* **Postcondizione:**

  L'utente visualizza correttamente tutti gli elementi ordinati rispetto al codice articolo.

UC 6.2 - Ordinamento per codice fornitore

* **Attori primari:**

  – Utente.

* **Precondizione:**

  L'utente visualizza correttamente la lista degli ordini.

* **Scenario principale:**

  1. L'utente ordina la lista per codice fornitore.

* **Postcondizione:**

  L'utente visualizza correttamente tutti gli elementi ordinati rispetto al codice fornitore.

UC 6.3 - Ordinamento per data d'ordine

* **Attori primari:**

  – Utente.

* **Precondizione:**

  L'utente visualizza correttamente la lista degli ordini.

* **Scenario principale:**

  1. L'utente ordina la lista per data d'ordine.

* **Postcondizione:**

  L'utente visualizza correttamente tutti gli elementi ordinati rispetto alla data d'ordine.

UC 6.4 - Ordinamento per data iniziale di copertura

* **Attori primari:**

  – Utente.

* **Precondizione:**

  L'utente visualizza correttamente la lista degli ordini.

* **Scenario principale:**

  1. L'utente filtra uno o più ordini per data iniziale di copertura.

* **Postcondizione:**

  L'utente visualizza correttamente tutti gli elementi ordinati rispetto alla data iniziale di copertura.

UC 6.5 - Ordinamento per data finale di copertura

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente la lista degli ordini.
* **Scenario principale:**
  1. L'utente filtra uno o più ordini per data finale di copertura.
* **Postcondizione:**

  L'utente visualizza correttamente tutti gli elementi ordinati rispetto alla data finale di copertura.

UC 6.6 - Ordinamento per quantità ordinata

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente la lista degli ordini.
* **Scenario principale:**
  1. L'utente filtra uno o più ordini per quantità ordinata.
* **Postcondizione:**

  L'utente visualizza correttamente tutti gli elementi ordinati rispetto alla quantità ordinata.

UC 6.7 - Ordinamento per prezzo parziale

* **Attori primari:**
  - Utente.
* **Precondizione:**

  L'utente visualizza correttamente la lista degli ordini.
* **Scenario principale:**
  1. L'utente ordina gli elementi rispetto al prezzo parziale.
* **Postcondizione:**

  L'utente visualizza correttamente tutti gli elementi ordinati rispetto al prezzo parziale.

3.2 Tracciamento dei requisiti

Da un'attenta analisi dei requisiti e degli use case effettuata sul progetto è stata stilata la tabella che traccia i requisiti in rapporto agli use case.
Sono stati individuati diversi tipi di requisiti e si è dunque utilizzato un codice identificativo univoco per distinguerli.
Il codice dei requisiti è così strutturato:

R<NumeroRequisito>-<Tipo>-<Classificazione>

In particolare il tipo può assumere 4 valori, quali:

* **F** = funzionale
* **Q** = qualitativo
* **P** = performance
* **V** = vincolo

Per quanto riguarda la classificazione, invece, si hanno 3 valori possibili:

* **O** = obbligatorio
* **D** = desiderabile
* **F** = facoltativo

Nelle tabelle 3.1, 3.2, 3.3 3.4 suddivise per tipo sono riassunti i requisiti e il loro tracciamento con gli use case delineati in fase di analisi.

Tabella 3.1: Tabella del tracciamento dei requisiti funzionali

Requisito	Descrizione	Use Case
R1-F-O	L'utente deve poter inserire i dati necessari per l'ottimizzazione	UC1
R2-F-O	L'utente deve poter inserire la data di inizio previsione	UC1.1
R3-F-O	L'utente deve poter inserire la data di fine previsione	UC1.2
R4-F-O	L'utente deve poter essere avvisato dell'errore di inserimento della data di inizio previsione	UC1.3
R5-F-O	L'utente deve poter essere avvisato dell'errore di inserimento della data di fine previsione	UC1.4
R6-F-O	L'utente deve poter iniziare l'analisi di ottimizzazione	UC2
R7-F-O	L'utente deve poter visualizzare i risultati	UC3
R8-F-O	L'utente deve poter visualizzare il totale non ottimizzato	UC3.1
R9-F-O	L'utente deve poter visualizzare il totale ottimizzato	UC3.2
R10-F-O	L'utente deve poter visualizzare lo scostamento tra i totali	UC3.3
R11-F-O	L'utente deve poter visualizzare la lista degli ordini in maniera decrescente rispetto al codice articolo	UC4

R12-F-O	L'utente deve poter visualizzare un singolo ordine della lista	UC4.1
R13-F-O	L'utente deve poter visualizzare il codice articolo di un ordine	UC4.1.1
R14-F-O	L'utente deve poter visualizzare il codice fornitore di un ordine	UC4.1.2
R15-F-O	L'utente deve poter visualizzare la data d'ordine di un ordine	UC4.1.3
R16-F-O	L'utente deve poter visualizzare la data iniziale di copertura di un ordine	UC4.1.4
R17-F-O	L'utente deve poter visualizzare la data finale di copertura di un ordine	UC4.1.5
R18-F-O	L'utente deve poter visualizzare la quantità ordinata di un ordine	UC4.1.6
R19-F-O	L'utente deve poter visualizzare il prezzo parziale di un ordine	UC4.1.7
R20-F-O	L'utente deve poter filtrare la lista	UC5
R21-F-O	L'utente deve poter filtrare la lista tramite una ricerca generica	UC5.1
R22-F-O	L'utente deve poter filtrare la lista per codice articolo	UC5.2
R23-F-O	L'utente deve poter filtrare la lista per codice fornitore	UC5.3
R24-F-O	L'utente deve poter filtrare la lista per data d'ordine	UC5.4
R25-F-O	L'utente deve poter filtrare la lista per data iniziale di copertura	UC5.5
R26-F-O	L'utente deve poter filtrare la lista per data finale di copertura	UC5.6
R27-F-O	L'utente deve poter filtrare la lista per quantità ordinata	UC5.7
R28-F-O	L'utente deve poter filtrare la lista per prezzo parziale	UC5.8
R29-F-O	L'utente deve poter ordinare la lista degli ordini	UC6
R30-F-O	L'utente deve poter ordinare la lista rispetto al codice articolo	UC6.1
R31-F-O	L'utente deve poter ordinare la lista rispetto al codice fornitore	UC6.2
R32-F-O	L'utente deve poter ordinare la lista rispetto alla data d'ordine	UC6.3
R33-F-O	L'utente deve poter ordinare la lista rispetto alla data iniziale di copertura	UC6.4
R34-F-O	L'utente deve poter ordinare la lista rispetto alla data finale di copertura	UC6.5

R35-F-O	L'utente deve poter ordinare la lista rispetto alla quantità ordinata	UC6.6
R36-F-O	L'utente deve poter ordinare la lista rispetto al prezzo parziale	UC6.7

Tabella 3.2: Tabella del tracciamento dei requisiti qualitativi

Requisito	Descrizione	Use Case
R37-Q-O	Deve essere redatto un documento che descrive l'architettura del modulo	-
R38-Q-O	Deve essere redatto un documento che spieghi le scelte implementative effettuate	-
R39-Q-O	Il codice deve essere documentato tramite commenti	-
R40-Q-D	L'algoritmo finale scelto deve generare dei log di chiamata per manutenzioni future	-
R41-Q-D	L'algoritmo di ottimizzazione deve essere estensibile	-
R42-Q-O	I test devono coprire il 60% del codice	-
R43-Q-D	L'algoritmo utilizza differenti tecniche di ottimizzazione	-
R44-Q-F	L'algoritmo utilizza il multithreading per cercare più soluzioni ammissibili	-

Tabella 3.3: Tabella del tracciamento dei requisiti di performance

Requisito	Descrizione	Use Case
R45-P-O	L'algoritmo di ottimizzazione deve restituire un risultato entro 10 minuti dal tempo di lancio dello stesso	-

Tabella 3.4: Tabella del tracciamento dei requisiti di vincolo

Requisito	Descrizione	Use Case
R46-V-O	La *form* deve essere eseguita sull'ambiente di esecuzione *.NET Framework*	-
R47-V-O	La *form* e l'algoritmo devono essere codificate in *C#*	-
R48-V-O	La versione utilizzata di *C#* deve essere 7.3	-

R49-V-O	La versione utilizzata di *.NET Framework* deve essere 4.8	-
R50-V-O	L'algoritmo finale deve fonire una soluzione ammissibile	-

Capitolo 4

Progettazione e codifica

Breve introduzione al capitolo

4.1 Tecnologie e strumenti

Di seguito viene data una panoramica delle tecnologie e strumenti utilizzati.

Tecnologia 1

Descrizione Tecnologia 1.

Tecnologia 2

Descrizione Tecnologia 2

4.2 Progettazione

Namespace 1

Descrizione namespace 1.

Classe 1: Descrizione classe 1

Classe 2: Descrizione classe 2

4.3 Design Pattern utilizzati

4.4 Codifica

Capitolo 5

Verifica e validazione

5.1 Verifica

5.1.1 Documentazione

5.1.2 Testing del modulo

5.2 Validazione

5.2.1 Documentazione

5.2.2 Codice

Capitolo 6

Conclusioni

6.1 Prodotto finale

6.2 Raggiungimento degli obiettivi

6.3 Conoscenze acquisite

Appendice A

Appendice A

Citazione

Autore della citazione

Glossario

.NET Framework Ambiente di esecuzione runtime della piattaforma tecnologica .NET in cui vengono gestite le applicazioni destinate allo stesso .NET Framework. Disponibile solo su *Windows*. 2

DevExpress Framework utile per lo sviluppo di applicazioni desktop. 2, 4

ERP Tipologia di software che integra tutti i processi di business rilevanti di un'azienda e tutte le funzioni aziendali, ad esempio vendite, acquisti, gestione magazzino, finanza, contabilità... 1

Server Consolidation È un approccio all'utilizzo efficiente delle risorse dei server dei computer al fine di ridurre il numero totale di server o posizioni di server richiesti da un'organizzazione. 1

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. 13

Bibliografia

Riferimenti bibliografici

James P. Womack, Daniel T. Jones. *Lean Thinking, Second Editon*. Simon & Schuster, Inc., 2010.

Siti web consultati

Manifesto Agile. URL: http://agilemanifesto.org/iso/it/.