

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 triennale

Relatore

Prof. Luigi De Giovanni

Laureando

Filippo Brugnolaro

Matricola 1217321

*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

1	Introduzione	1
1.1	L'azienda	1
1.2	L'idea	2
1.3	Descrizione dello <i>stage</i>	2
1.3.1	Introduzione	2
1.3.2	Obiettivi	4
1.3.3	Analisi preventiva dei rischi	4
1.4	Organizzazione del testo	5
2	Studio di fattibilità	7
2.1	Introduzione allo studio	7
2.2	Soluzioni proposte	8
2.2.1	Algoritmo <i>Greedy</i>	8
2.2.2	<i>Tabu search</i>	9
2.2.3	<i>Algoritmo genetico</i>	10
2.3	Conclusioni dello studio	12
3	Analisi dei requisiti	13
3.1	Casi d'uso	13
3.2	Tracciamento dei requisiti	28
4	Progettazione e codifica	33
4.1	Architettura	33
4.2	Funzionamento generale	34
4.3	<i>Tabu search</i>	35
4.3.1	Rappresentazione della soluzione	36
4.3.2	Soluzione iniziale	36
4.3.3	Mosse	37
4.3.4	Esplorazione del vicinato	37
4.3.5	<i>Tabu List</i>	38
4.3.6	Funzione di valutazione	38
4.3.7	Condizioni di arresto	40
4.3.8	Controllo dei vincoli	40
4.4	Codifica	42
4.4.1	Organizzazione dello sviluppo	42
4.4.2	<i>Log</i>	42
4.4.3	Estensioni del progetto	43
4.5	Tecnologie e strumenti	44
5	Verifica e validazione	47
5.1	Verifica	47
5.1.1	Modalità	47
5.1.2	<i>Testing</i> del modulo	47

5.2	Validazione	48
5.2.1	Codice	48
5.2.2	Requisiti	48
5.2.3	Risultati dei <i>test</i>	51
6	Conclusioni	55
6.1	Prodotto finale	55
6.2	Raggiungimento degli obiettivi	55
6.3	Conoscenze acquisite	55
6.4	Valutazione complessiva	56
	Glossario	57
	Bibliografia	61

Elenco delle figure

1.1	Logo Ergon Informatica S.R.L.	1
3.1	Use case - sistema principale	13
3.2	UC1 - Inserimento dati	14
3.3	UC3 - Visualizzazione risultati	16
3.4	UC4 - Visualizzazione lista degli ordini	18
3.5	UC4.1 - Visualizzazione singolo ordine	18
3.6	UC 5 - Filtraggio dati	21
3.7	UC 6 - Ordinamento della lista degli ordini	25
4.1	Architettura generale	33
4.2	Diagramma di attività della <i>Windows Form</i>	34
4.3	Diagramma di attività della <i>Tabu search</i>	35
4.4	Rappresentazione della soluzione per la <i>Tabu search</i>	36
4.5	Logo <i>C#</i>	44
4.6	Logo <i>Visual Studio 2019</i>	44
4.7	Logo <i>DevExpress</i>	45
4.8	Logo <i>IBM Informix</i>	45
4.9	Logo <i>Git</i>	45
5.1	Grafico scostamento/iterazioni	52
5.2	Grafico scostamento/risoluzione	53

Elenco delle tabelle

1.1	Esempio - Fabbisogni	3
1.2	Esempio - Listino Prezzi	3
1.3	Esempio - Calendario spedizioni	3
2.1	Tabella dei risultati dell'analisi degli algoritmi dopo 10 esecuzioni	12
3.1	Tabella del tracciamento dei requisiti funzionali	28
3.2	Tabella del tracciamento dei requisiti qualitativi	30
3.3	Tabella del tracciamento dei requisiti di performance	31
3.4	Tabella del tracciamento dei requisiti di vincolo	31

3.5	Riepilogo dei requisiti	31
4.1	Tabella dei risultati di f	39
4.2	Tabella dei risultati di g	39
4.3	Tabella dei risultati di h	39
5.1	Tabella della validazione dei requisiti funzionali	48
5.2	Tabella della validazione dei requisiti qualitativi	50
5.3	Tabella della validazione dei requisiti di performance	50
5.4	Tabella della validazione dei requisiti di vincolo	51
5.5	<i>Test</i> - variazione del numero massimo di iterazioni	51
5.6	<i>Test</i> - scelta di risoluzione dei minimi	52
6.1	Riepilogo della validazione dei requisiti	55

Elenco degli algoritmi

1	Pseudocodice <i>string replacement</i> - Algoritmo <i>greedy</i>	8
2	Pseudocodice <i>string replacement</i> - <i>Tabu search</i>	9
3	Pseudocodice <i>string replacement</i> - Algoritmo <i>genetico</i>	10
4	Pseudocodice soluzione iniziale - Algoritmo <i>greedy</i>	36

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 Informatica*") 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*.

¹Sito 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 | * <i>Web</i> |
| * Controllo di Gestione | * <i>Business Intelligence</i> |
| * Area Acquisti | * Qualità |
| * Logistica | * Gestione Archivi e Documentazione |
| * Vendite | * Pianificazione Consegne |
| * Produzione | * <i>Area Mobile</i> |

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 nello 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 il modulo attuale, per ogni prodotto da ordinare, prende 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 nuovo 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 a non considerare:

- * **l'andamento del mercato:** i prezzi sono variabili e dipendono dal periodo nel quale si comprano i prodotti.
- * **i 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 sarebbe 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 *Windows form* 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 delle 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 di valutazione

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

- * definire una funzione di valutazione 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 approfondisce 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 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 effettuare 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:** quantitativo di risorse temporali impiegate per sviluppare l'algoritmo.
- * **Paper:** dichiarazioni o dati di esperimenti già effettuati.

²Verranno 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 predefinito, 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  $\leftarrow$  generate_array(stringa_errata)
3:   array_str_corr  $\leftarrow$  generate_array(stringa_corretta)
4:   funzione_obiettivo  $\leftarrow$  array_str_corr.Length
5:   pos  $\leftarrow$  0
6:   for each element  $\in$  array_str_err do
7:     if element  $\neq$  array_str_corr[pos] then
8:       element = array_str_corr[pos]
9:     end if
10:    funzione_obiettivo  $\leftarrow$  funzione_obiettivo - 1
11:    pos  $\leftarrow$  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* (*Metodo a traiettoria*), 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  $\leftarrow$  stringa_errata  
3:   funzione_obiettivo_curr  $\leftarrow$  stringa_corretta.Length  
4:   funzione_obiettivo_best  $\leftarrow$  funzione_obiettivo_curr  
5:   conta  $\leftarrow$  0  
6:   while conta < max_iterazioni do  
7:     vicinato  $\leftarrow$  genera_vicinato()  
8:     while vicinato.Length > 0 and conta < max_iterazioni do  
9:       vicino_migl  $\leftarrow$  ottieni_miglior_vicino(vicinato)  
10:      if vicino_migl  $\notin$  tabu_list then  
11:        funzione_obiettivo_curr  $\leftarrow$  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  $\leftarrow$  vicino_migl  
15:          funzione_obiettivo_best  $\leftarrow$  funzione_obiettivo_curr  
16:          conta  $\leftarrow$  conta + 1  
17:          break  
18:        end if  
19:      end if  
20:      Rimuovi vicino_migl dal vicinato  
21:      conta  $\leftarrow$  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 appartenenza 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.

³Per una descrizione più accurata si legga la sezione [4.4](#)

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, classificato 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 appartenente 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:   arrstr_corr ← codifica(stringa_corretta)  
3:   lista_pop ← inizializza_pop()  
4:   set_fitness()  
5:   fitnessbest ← most_fitness_val(arrstr_corr)  
6:   solbest ← most_fitness_sol(arrstr_corr)  
7:   numcrossover ← calculate_num_crossover(crossover_rate, lista_pop)  
8:   conta ← 0
```

⁴Scienziato 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.

```

9:  while conta < max_iterazioni do
10:     lista_pop ← seleziona_individui(num_crossover)
11:     set_fitness()
12:     i ← 0
13:     while i < num_crossover do
14:         lista_genitori ← seleziona_individui(2)
15:         figlio ← genera_figlio(lista_genitori)
16:         Aggiungi il figlio in coda alla lista nuova_pop
17:         i ← i + 1
18:     end while
19:     lista_pop ← nuova_pop
20:     lista_pop ← mutazione_rand(lista_pop)
21:     set_fitness()
22:     fitness_curr ← most_fitness_val(arr_str_corr)
23:     sol_curr ← most_fitness_sol(arr_str_corr)
24:     if fitness_curr > fitness_best then
25:         fitness_best ← fitness_curr
26:         sol_best ← sol_curr
27:     end if
28:     conta ← conta + 1
29: end while
30: return sol_best, fitness_best
31: 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;
- * 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.

Tabella 2.1: Tabella dei risultati dell'analisi degli algoritmi dopo 10 esecuzioni

Tipologia	Efficacia (%)	Efficienza (ms)	Tempo di realizzazione (h)
Algoritmo <i>greedy</i>	100	0,45	0,5
<i>Tabu search</i>	98	1289,76	2,5
Algoritmo Genetico	92	5127,24	5

L'algoritmo *greedy* risulta il più efficiente ed efficace in questo specifico caso, ma per il problema che si andrà a risolvere non può essere considerato molto soddisfacente in quanto le scelte non vengono mai rimesse in discussione e potrebbero dunque escludere soluzioni potenzialmente migliori (esempio nella sezione §1.3.1).

Per quanto riguarda l'algoritmo genetico, sebbene fosse interessante per il mantenimento di più soluzioni ad ogni iterazione, si è dimostrato molto più lento rispetto ai due precedenti. Inoltre una maggiore complessità realizzativa lo ha portato all'esclusione.

Si è optato quindi 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 può essere generata a partire da un algoritmo *greedy*, data la sua bassa complessità computazionale. In questo modo è possibile avere già a disposizione una buona soluzione di base. Tuttavia, sebbene la *Tabu search* possieda una sola soluzione ad ogni iterazione, è stato pensato che l'algoritmo possa essere lanciato in più *thread*, riuscendo dunque in un certo senso a simulare il mantenimento di più soluzioni dell'algoritmo genetico, passando ai *thread* chiaramente soluzioni iniziali differenti.

Capitolo 3

Analisi dei requisiti

In questo capitolo viene spostata l'analisi dei requisiti effettuata durante lo stage, nella quale vengono descritte le funzionalità tramite i casi d'uso.

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.

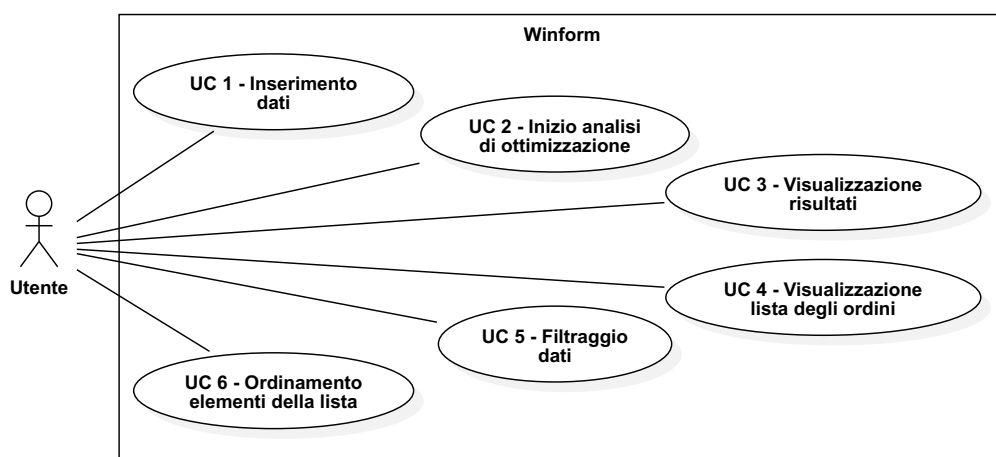


Figura 3.1: Use case - sistema principale

UC 1 - Inserimento dati

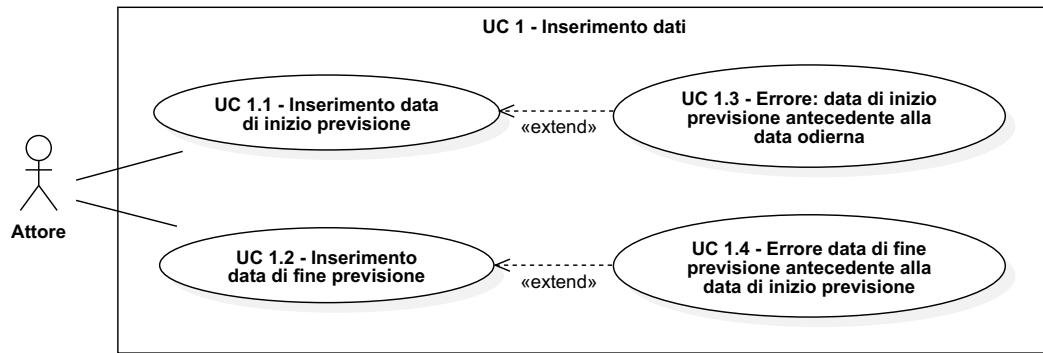


Figura 3.2: UC1 - Inserimento dati

* **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

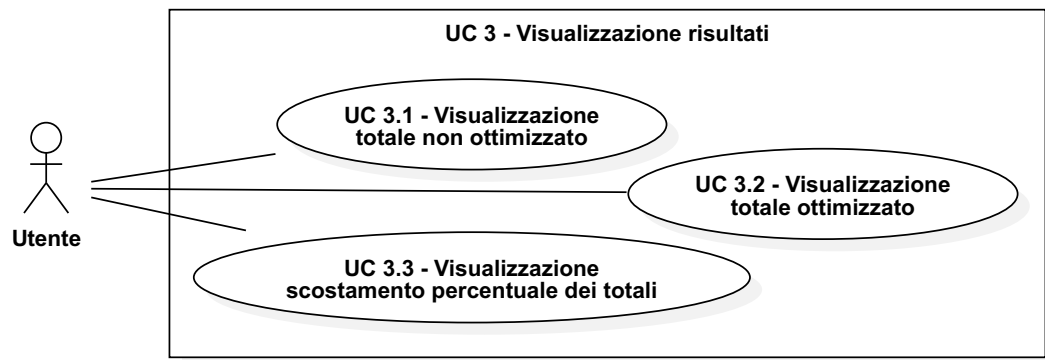


Figura 3.3: UC3 - Visualizzazione risultati

* **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

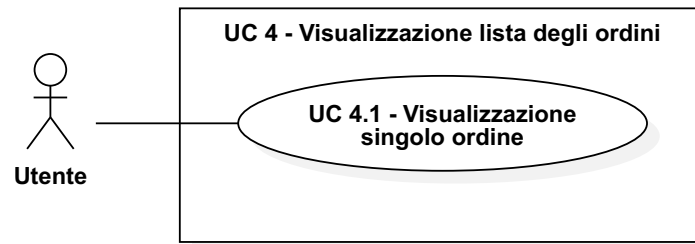


Figura 3.4: UC4 - Visualizzazione lista degli ordini

* **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

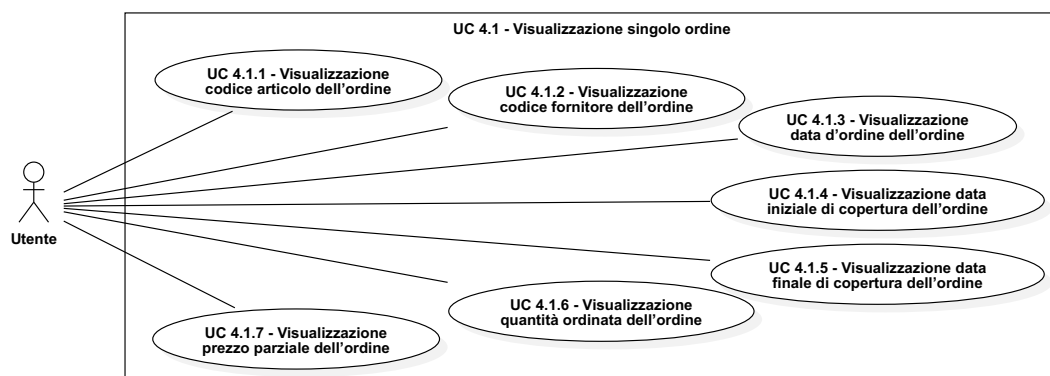


Figura 3.5: UC4.1 - Visualizzazione singolo ordine

* **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

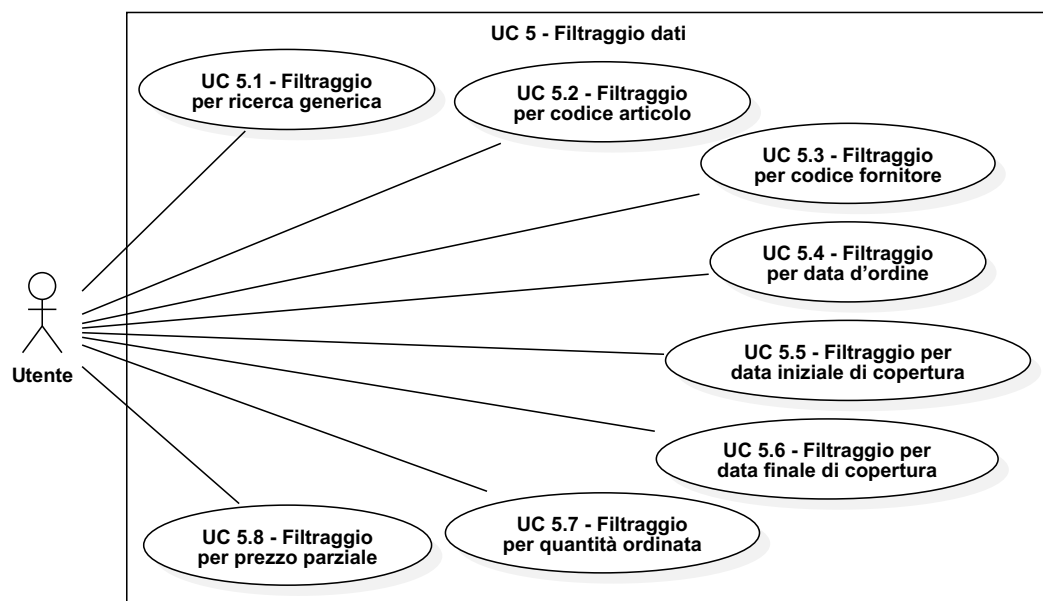


Figura 3.6: UC 5 - Filtraggio dati

* **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

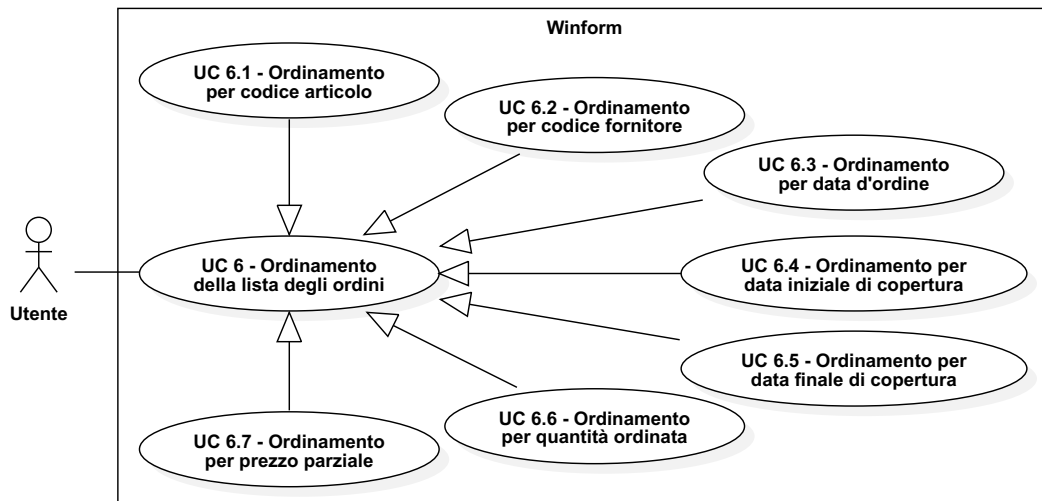


Figura 3.7: UC 6 - Ordinamento della lista degli ordini

* **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:

$$\mathbf{R}<\mathbf{NumeroRequisito}>-<\mathbf{Tipo}>-<\mathbf{Classificazione}>$$

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 e 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 <i>log</i> di chiamata per manutenzioni future	-
R41-Q-D	L'algoritmo di ottimizzazione deve essere estensibile	-
R42-Q-D	I <i>test</i> devono coprire il 60% del codice	-
R43-Q-D	L'algoritmo utilizza differenti tecniche di ottimizzazione	-
R44-Q-F	L'algoritmo utilizza il <i>multithreading</i> 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 <i>form</i> deve essere eseguita sull'ambiente di esecuzione <i>.NET Framework</i>	-
R47-V-O	La <i>form</i> e l'algoritmo devono essere codificate in <i>C#</i>	-
R48-V-O	La versione utilizzata di <i>C#</i> deve essere 7.3	-
R49-V-O	La versione utilizzata di <i>.NET Framework</i> deve essere 4.8	-
R50-V-O	L'algoritmo finale deve fornire una soluzione ammissibile	-

Tabella 3.5: Riepilogo dei requisiti

Tipo	Obbligatori	Desiderabili	Facoltativi
Funzionali	36	0	0
Qualitativi	3	4	1
Prestazionali	1	0	0
Vincolo	5	0	0

Capitolo 4

Progettazione e codifica

In questo capitolo vengono esposte le attività di progettazione e codifica del modulo di ottimizzazione.

4.1 Architettura

Prima di descrivere più in particolare l'algoritmo, è necessario fornire un'idea dell'architettura sulla quale si basa l'intero progetto. Di seguito è presentato uno schema ad alto livello di come sono strutturate le varie componenti che formano il sistema.

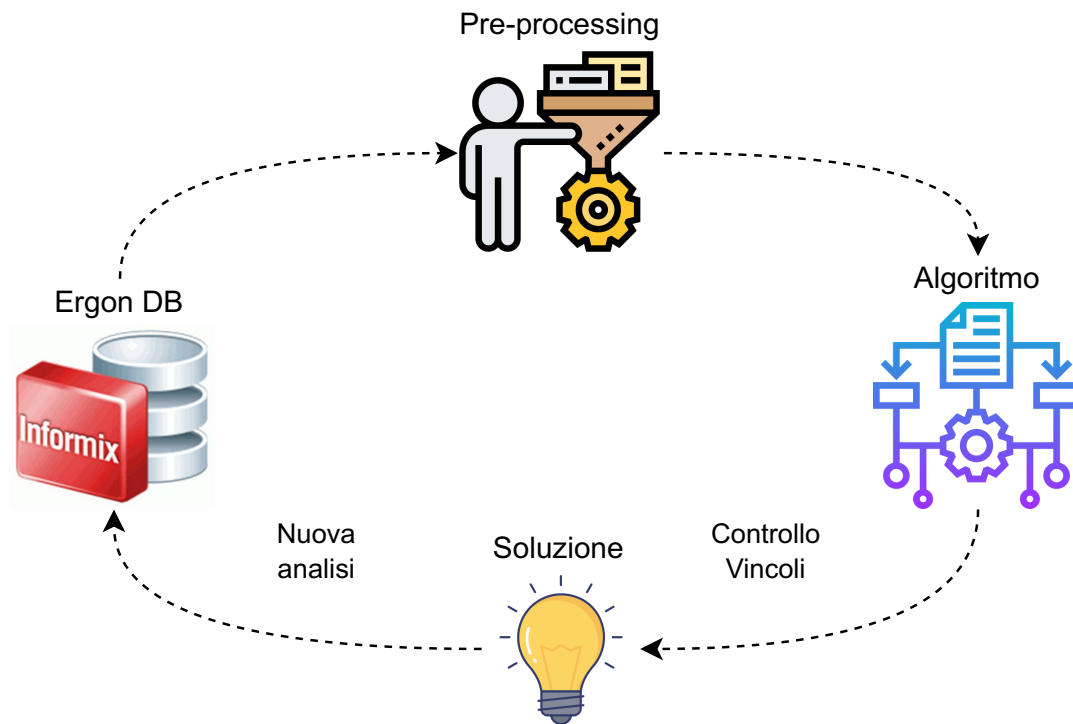


Figura 4.1: Architettura generale

Come si può notare in figura 4.1, il flusso del sistema è circolare. Si parte dal *database*, dal quale vengono estratti i dati di interesse dalle varie tabelle. Successivamente viene fatto il *pre-processing* dei dati, in modo tale da poter poi eseguire l'ottimizzazione solo su ciò che è di interesse dell'analisi. Di seguito viene effettuato il controllo dei vincoli di minimo al termine dell'algoritmo così da generare una soluzione *ammissibile*. Se si vuole effettuare una nuova analisi è necessario estrarre nuovamente i dati dal *database* e rieseguire il ciclo. Questo è necessario in quanto i dati all'interno del *database* possono variare (esempio: prezzi, date di spedizione...).

4.2 Funzionamento generale

In questa sezione viene descritta una classica interazione dell'utente con il programma. Nella figura 4.2 si riassume il flusso delle interazioni.

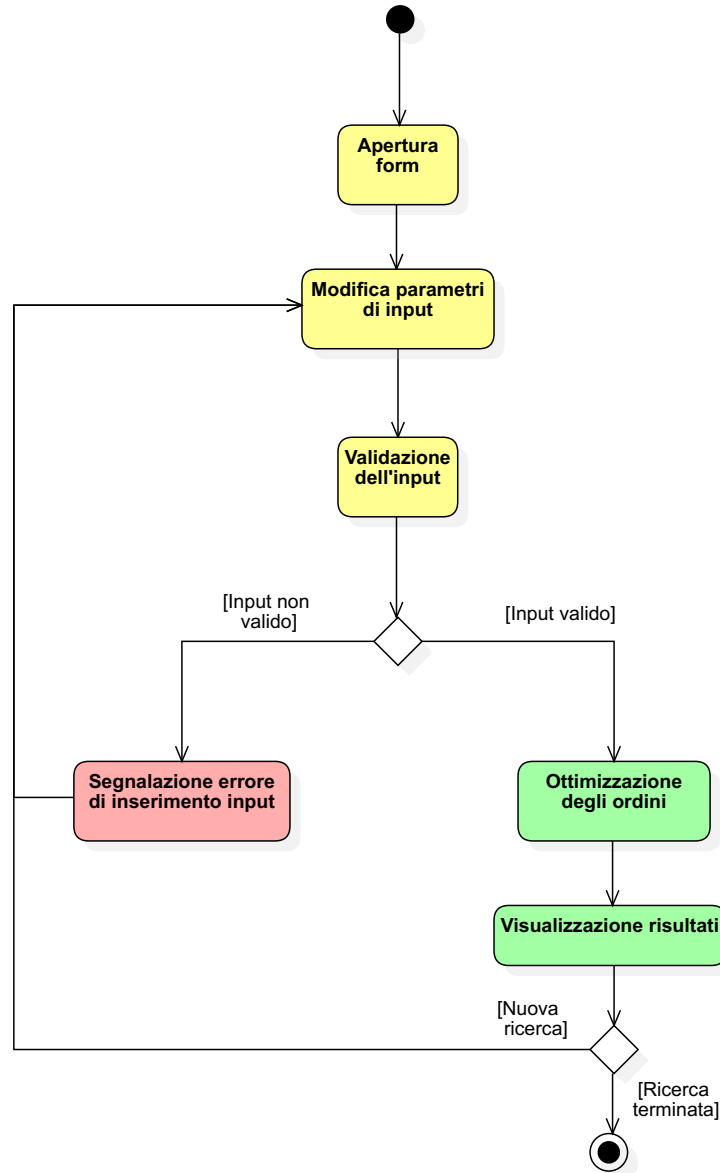


Figura 4.2: Diagramma di attività della *Windows Form*

L'utente, all'apertura della *Windows form*, modifica i parametri di *input* che sono:

- * data di previsione iniziale
- * data di previsione finale
- * metodo di risoluzione dei vincoli

Dopo aver scelto gli *input*, questi vengono validati in modo tale da bloccare anzitempo l'esecuzione in caso di errori. Se la validazione va a buon fine, allora l'algoritmo calcola la soluzione e vengono visualizzati i risultati. A questo punto la ricerca può terminare oppure può continuare con la possibilità di variare i parametri di *input*.

4.3 *Tabu search*

Nella sezione § 2.3 riguardante lo studio di fattibilità è emerso come la *Tabu search* sia il giusto compromesso in termini di efficacia, efficienza e complessità a livello implementativo. In figura 4.3 viene rappresentato il funzionamento dell'algoritmo ad alto livello.

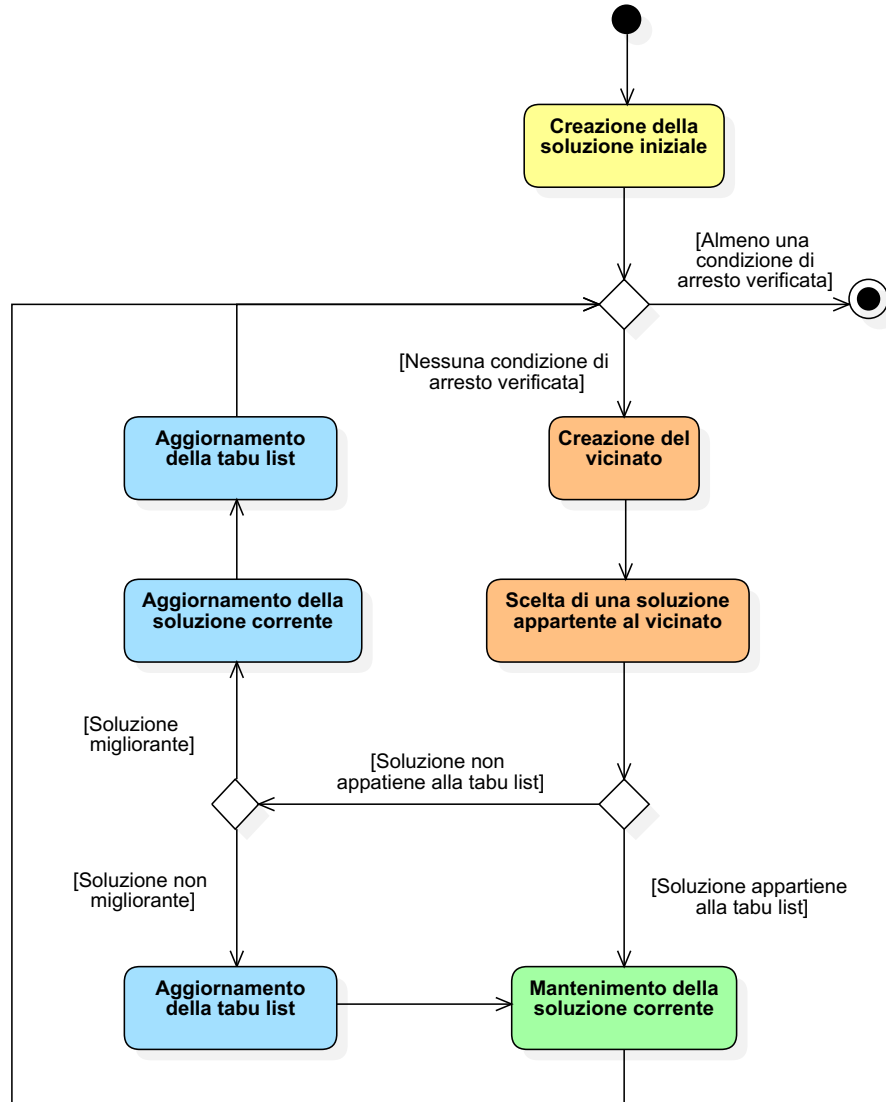


Figura 4.3: Diagramma di attività della *Tabu search*

Come si può notare, prima viene effettuato un controllo per vedere se la mossa appartiene o meno alla *Tabu list* e poi viene verificato se la soluzione è migliorante. Se i controlli venissero invertiti, si rischierebbe, calcolando la funzione di valutazione e confrontandola con quella della soluzione corrente, di simulare inutilmente una mossa che potenzialmente potrebbe appartenere alla *Tabu list*.

4.3.1 Rappresentazione della soluzione

Definire una buona rappresentazione della soluzione è fondamentale poichè è legata alla creazione del vicinato.

In primis è stata creata una classe che rappresentasse il singolo ordine (*dataSourceItem*) e, dato che gli ordini solitamente sono molteplici, si è deciso di mettere ognuno di essi all'interno di una lista. Questo approccio è sufficiente per un algoritmo *greedy* ma non per la *Tabu search*.

Infatti è necessario associare alla soluzione anche una funzione di valutazione, in modo tale da poter operare il confronto tra la soluzione corrente e quella migliore.

In figura 4.4 viene illustrata la forma della soluzione della *Tabu search*.

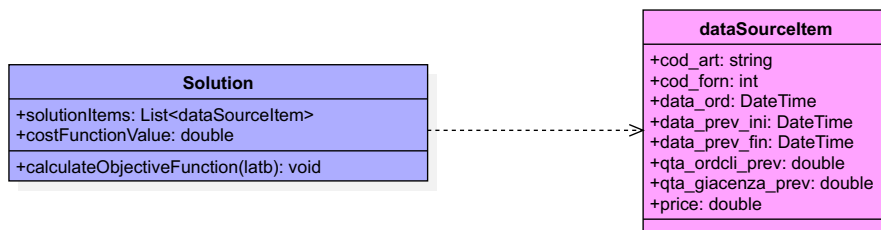


Figura 4.4: Rappresentazione della soluzione per la *Tabu search*

4.3.2 Soluzione iniziale

Nella figura 4.3 il primo step è la creazione di una soluzione iniziale che però la *Tabu search*, a contrario di altri algoritmi, non è in grado di generare da sola. Per ovviare a questo inconveniente si può generare una soluzione abbastanza buona tramite un algoritmo *greedy*. Le scelte *greedy* implementate sono:

- * **HighestArtRequestByArt:** si sceglie il codice articolo più richiesto e viene comprato ogni *range* di copertura riferito all'articolo stesso;
- * **HighestArtRequestByForn:** si sceglie l'articolo più richiesto, il fornitore che offre il prezzo minore e vengono comprati tutti i *range* di copertura di ogni articolo da quel fornitore;
- * **HighestVariationByArt:** si sceglie l'articolo con più variazione del prezzo unitario rispetto a quello dell'anno precedente e viene comprato ogni *range* di copertura riferito all'articolo stesso;
- * **HighestVariationByForn:** si sceglie l'articolo con più variazione del prezzo unitario rispetto a quello dell'anno precedente, il fornitore che offre il prezzo minore e vengono comprati tutti i *range* di copertura di ogni articolo da quel fornitore.

Di seguito viene presentato lo pseudocodice.

Pseudocodice soluzione iniziale - Algoritmo *greedy*

input: *lista_articoli*, *lista_spedizioni*, *lista_prezzi*

Output: *lista_articoli_opt*

- 1: **procedure** MY_GREEDY_ALGORITHM(*lista_articoli*, *lista_spedizioni*, *lista_prezzi*)
 - 2: *l* ← *lista_vuota*
 - 3: *l_{tmp}* ← *lista_vuota*
 - 4: *l_{cod_art}* ← *genera_lista_decisione(lista_articoli)*
 - 5: *n_{cod_art}* ← *l_{cod_art}.Length*
-

```

6:  while  $count < n_{cod\_art}$  do
7:     $cod\_art \leftarrow decision\_greedy(l_{cod\_art})$ 
8:    for each  $art \in lista\_articoli$  do
9:       $b \leftarrow controllo\_condizioni(art, cod\_art, lista\_spedizioni, lista\_prezzi)$ 
10:     if  $b = true$  then
11:       Aggiungi  $art$  a  $l_{tmp}$ 
12:     end if
13:   end for
14:    $count \leftarrow count + 1$ 
15: end while
16:  $l \leftarrow filtra\_articoli\_per\_min(l_{tmp})$ 
17: return  $l$ 
18: end procedure

```

Si può notare come la funzione *controllo_condizioni()* abbia lo scopo di escludere tutte le casistiche relative a tutti i *cod_art* in cui:

- * la data di spedizione è minore del giorno attuale;
- * la data di arrivo della spedizione è maggiore della data di inizio copertura;
- * dato un fornitore, non esiste un prezzo associato ad alcuna spedizione di *cod_art*;
- * dato un fornitore, non esiste alcuna spedizione associata a *cod_art*.

La funzione *filtra_articoli_per_min()*, avente come parametro di input l_{tmp} (lista con tutti i record che soddisfano i vincoli), serve invece a filtrare per prezzo minimo tutti i *range* di copertura associati ad ogni codice articolo. In pratica per tutti i *range* di copertura di ogni articolo viene preso il record con il fornitore che offre il fabbisogno al prezzo più vantaggioso.

È possibile avere anche un'inizializzazione della soluzione che non sia stata soggetta ad alcuna ottimizzazione. In questo caso è stata creata una funzione *initialise_solution()* che, dati in input gli stessi parametri dell'algoritmo *greedy*, fornisce come soluzione iniziale proprio quella generata dal modulo già esistente.

4.3.3 Mosse

Le mosse create per la *Tabu search* sono le seguenti:

1. Inserimento di un nuovo ordine d'acquisto;
2. Cambio di fornitore di un ordine d'acquisto;
3. Pre-ordine di un ordine d'acquisto;
4. Post-ordine di un ordine d'acquisto.

Come si può notare non è stata creata la mossa inversa dell'inserimento, ovvero la rimozione di un ordine d'acquisto. Il motivo sta nel fatto che rimuovere un ordine di un articolo di cui si ha bisogno non porta sicuramente a un miglioramento della funzione di valutazione poichè, essendo l'articolo una necessità, prima o poi verrà aggiunto nuovamente.

4.3.4 Esplorazione del vicinato

Il vicinato è quell'insieme di soluzioni che possono essere raggiunte tramite l'applicazione di una mossa sulla soluzione corrente (chiamata anche *centro del vicinato*).

Si può notare come la dimensione del vicinato in questo problema sia dell'ordine di $O(n!)$ e sia dettata dal fatto che per tutti i *range* di copertura di ogni articolo sia necessario scegliere la miglior combinazione tra data d'ordine e fornitore.

Data la dimensione importante del vicinato non è dunque possibile visitarlo per intero, motivo per il quale si è deciso di procedere randomizzando le scelte. In particolare ogni mossa viene scelta in maniera casuale cosicché tutte le mosse abbiano la stessa probabilità di essere scelte.

Per il medesimo motivo è stato applicato lo stesso principio anche per la scelta dell'articolo su cui applicare la mossa.

È importante sottolineare come l'oggetto *Random* in *C#* debba essere inizializzato tramite un numero, ovvero ciò che viene definito come *seed*. Infatti se non si dichiara il *seed* all'interno del costruttore, l'oggetto viene inizializzato usando il *Current System Time*. Questo risultava un problema poichè, essendo le iterazioni molto rapide (a volte anche $< 1ms$), si andava ad inizializzare la nuova istanza di *Random* allo stesso valore e dunque veniva generato il medesimo numero, andando a crearsi una situazione indesiderata.

4.3.5 *Tabu List*

La *Tabu list* è una lista che memorizza le k mosse precedenti, dove k è uguale alla lunghezza della lista che viene definita quando viene creato l'oggetto *TabuSearch*.

Permette di evitare dunque cicli infiniti in corrispondenza di minimi locali. Questo accade perché tramite la funzione *generate_move_string()* ogni mossa viene prima codificata e poi viene effettuata una ricerca nella *Tabu list*, impedendo potenzialmente all'algoritmo di riprovarla. Se la mossa non viene trovata, questa viene aggiunta nella *Tabu list* sia nel caso in cui l'operazione venga compiuta che nel caso in cui non venga compiuta perchè non migliorante.

Per semplificare la codifica si salva solamente la stringa generata dalla funzione che rappresenta la mossa e non l'intera soluzione come previsto dalla teoria. Si è deciso inoltre di memorizzare, oltre alla mossa eseguita, anche la sua inversa (per esempio l'inversa del pre-ordine è il post-ordine). Così facendo si evitano le inversioni immediate che risulterebbero poco sensate e rallenterebbero la ricerca.

4.3.6 Funzione di valutazione

I parametri da considerare per valutare quanto sia buona una soluzione, vista dal lato utente, sono i seguenti:

- * il numero di articoli ordinati rispetto a quelli totali;
- * il prezzo totale di tutti gli ordini effettuati.

Chiaramente la funzione di valutazione deve migliorare se aumenta il numero di articoli ordinati oppure se, a parità di articoli ordinati, il prezzo totale diminuisce. Al contrario si ha un peggioramento nel caso in cui, a parità di articoli ordinati, corrisponde un prezzo totale maggiore. Non si potrà mai avere invece un peggioramento della funzione di valutazione causato dalla diminuzione del numero di articoli ordinati in quanto la mossa è inesistente, come spiegato nella sezione §4.3.3. La funzione di valutazione, in questo caso, non può rappresentare perfettamente il comportamento dell'algoritmo e definisce dunque un'approssimazione del suo andamento.

Durante la ricerca sono state provate empiricamente 3 funzioni che vengono elencate qui sotto:

$$f = \frac{PT \cdot (1-R)}{e^R} \quad g = \begin{cases} \frac{\ln(PT) \cdot (1-R)}{e^R} & \text{se } R \neq 1 \\ -\frac{1}{PT} & \text{se } R = 1 \end{cases} \quad h = \begin{cases} \frac{\ln(PT+1) \cdot (1-R)}{e^R} & \text{se } R \neq 1 \\ -\frac{1}{PT} & \text{se } R = 1 \end{cases}$$

dove:

- * PT = prezzo totale
- * OE = numero di articoli ordinati
- * OT = numero di articoli da ordinare
- * $R = \frac{OE}{OT}$ = rapporto tra gli articoli ordinati e quelli da ordinare

Nelle tabelle 4.1, 4.2 e 4.3 vengono riportati i risultati che sono stati calcolati per capire empiricamente quale fosse la funzione di valutazione che approssimasse meglio il comportamento dell'algoritmo.

Tabella 4.1: Tabella dei risultati di f

PT	Rapporto $\frac{OE}{OT}$		FV
250.000	30/1300	0,0230	238.659,21
400.000	40/1300	0,0307	375.944,97
500.000	50/1300	0,0384	462.629,19
600.000	60/1300	0,0461	546.493,78
350.000	21/1300	0,0161	338.828,33
350.000	22/1300	0,0169	338.303,08
350.000	23/1300	0,0176	337.778,43
400.000	23/1300	0,0176	386.032,50
400.000	24/1300	0,0184	385.433,60
500.000	1300/1300	1,0000	0,00000000
450.000	1300/1300	1,0000	0,00000000

Tabella 4.2: Tabella dei risultati di g

PT	Rapporto $\frac{OE}{OT}$		FV
250.000	30/1300	0,0230	11,8653876
400.000	40/1300	0,0307	12,1234920
500.000	50/1300	0,0384	12,1415767
600.000	60/1300	0,0461	12,1182126
350.000	22/1300	0,0169	12,3390619
350.000	23/1300	0,0176	12,3199264
400.000	23/1300	0,0176	12,4487951
400.000	24/1300	0,0184	12,4294818
0,75	1/1300	0,0008	-0,2872397
500.000	1300/1300	1,0000	-0,0000020
450.000	1300/1300	1,0000	-0,0000022

Tabella 4.3: Tabella dei risultati di h

PT	Rapporto $\frac{OE}{OT}$		FV
250.000	30/1300	0,0230	11,8653914
400.000	40/1300	0,0307	12,1234943
500.000	50/1300	0,0384	12,1415785
600.000	60/1300	0,0461	12,1182141
350.000	22/1300	0,0169	12,3390647
350.000	23/1300	0,0176	12,3199292
400.000	23/1300	0,0176	12,4487975
400.000	24/1300	0,0184	12,4294842
0,75	1/1300	0,0008	0,55875534
500.000	1300/1300	1,0000	-0,0000020
450.000	1300/1300	1,0000	-0,0000022

È evidente dalla tabella 4.1 come la funzione f sia migliorativa se, a parità di PT , si aumenta il numero di articoli ordinati. Inoltre FV dovrebbe migliorare anche all'aumentare in contemporanea di R e PT poichè si andrebbe a coprire più fabbisogni, ma ciò non succede. Questo problema accade perchè il rapporto tra R e PT è troppo sbilanciato.

Altra problematica è il fatto che la funzione, una volta ordinati tutti gli articoli, non riesce a

valutare la variabile PT in quanto $1 - R$ è uguale a 0.

Si è optato quindi per la modifica della funzione di valutazione e si è ottenuta la funzione g .

Si osservi ora la tabella 4.2. Per cercare di risolvere il primo problema, si è deciso di adottare un compromesso andando ad utilizzare il logaritmo naturale.

Dal quinto all'ottavo risultato si può evincere che:

1. a parità di $PT \rightarrow FV$ è migliore per un R maggiore;
2. a parità di $R \rightarrow FV$ è migliore per un PT minore.

Tuttavia si può notare come non sia stato comunque completamente risolto il problema perchè nei primi tre risultati FV aumenta e solo nel quarto FV inizia a decrescere. Questo potrebbe sembrare un problema, ma in realtà non lo è in quanto la mossa disponibile nella *Tabu search* inserisce un solo elemento per volta riconducendoci quindi più verosimilmente alle casistiche dell'intervallo che va dal quinto all'ottavo risultato.

Per il secondo problema invece si è dovuto procedere per casi in base al valore assunto da PT . In questo modo, non appena tutti gli articoli sono stati ordinati ($R = 1$), è possibile operare un confronto. Data dunque FV , definita come il negativo del reciproco di PT , è facile vedere come al diminuire di PT diminuisca anche FV , risolvendo dunque completamente il problema.

Il decimo risultato è stato determinante poichè ha permesso di scoprire un altro problema che è principalmente di dominio. Infatti la funzione g valuta tutti i risultati con $0 < PT < 1$ e $R < 1$ come miglioranti persino rispetto a quei risultati con $R = 1$. Per risolvere questo problema si è andati a rimodificare la funzione di valutazione ottenendo h .

Si osservi ora la tabella 4.3. Per risolvere il problema esposto precedentemente si è andato a modificare il dominio della funzione aggiungendo una unità. In questo modo per ogni valore di PT si avrà che $\ln(PT + 1) > 0$ facendo in modo che FV sia negativa se e solo se $R = 1$.

4.3.7 Condizioni di arresto

La *Tabu search* si ferma se si verifica almeno una delle seguenti condizioni:

- * **Numero massimo di iterazioni:** numero di mosse eseguibili dall'algoritmo, deciso dal programmatore, che permette di evitare la possibilità di eventuali cicli infiniti;
- * **Numero massimo di iterazioni non migliorative:** numero di mosse non migliorative consecutive eseguibili dall'algoritmo, deciso dal programmatore, che permette di rilevare in maniera approssimativa un minimo locale e terminare la procedura;
- * **Tempo massimo di esecuzione:** durata temporale oltre la quale il programma si ferma automaticamente. Questo serve per venire incontro a bisogni di rapidità di risposta da parte dell'azienda.

4.3.8 Controllo dei vincoli

Nel programma i vincoli vengono sempre controllati alla fine dell'esecuzione dell'algoritmo tramite due funzioni:

- * *control_Min_Order_Articles()*, con *input* la lista di ordini effettuati e la lista dei *bound* rispetto al singolo ordine;
- * *control_Mins_Forn()*, con *input* la lista di ordini effettuati, la lista delle date di spedizione, la lista dei *bound* dei fornitori e la scelta della modalità di risoluzione dei vincoli.

La prima funzione controlla per tutti *range* di copertura di ogni articolo che l'ordine:

1. rispetti il quantitativo minimo da ordinare in quella specifica data d'ordine;
2. venga effettuato per multipli se la quantità dell'ordine supera il quantitativo minimo da ordinare.

Il primo vincolo viene risolto aggiungendo la differenza alla quantità attuale in modo tale da arrivare al minimo.

Il secondo, invece, viene risolto in questo modo:

1. si trova la differenza tra la quantità attuale e quella minima richiesta
2. viene eseguito il modulo tra la differenza calcolata al punto 1 e il valore del multiplo
3. viene aggiunta alla quantità attuale la differenza tra il valore del multiplo e il valore calcolato al punto precedente

La seconda funzione, invece, controlla per ogni periodo e per ogni fornitore a cui si abbia fatto almeno un ordine che:

- * il numero degli articoli totali comprati all'interno del determinato periodo sia maggiore del *bound*
- * l'importo totale all'interno del periodo sia maggiore del *bound*

Per il soddisfacimento di questi due vincoli, l'utente ha la possibilità di scegliere quale tra le seguenti tecniche adottare:

1. **Min/Max:** per il primo (risp. secondo) vincolo si deve scegliere l'articolo minimo (risp. massimo) appartenente al periodo in questione, che verrà poi aumentato fino ad arrivare al minimo dichiarato dal *bound*.
2. **Proporzionale:** consiste nel continuare a ciclare tutti gli articoli appartenenti al periodo e aumentarli di una unità finché non si è raggiunto il minimo dichiarato dal *bound*.
3. **Ricompattamento:** consiste nel selezionare l'ordine successivo con lo stesso fornitore, con il prezzo minimo e non appartenente al periodo in modo tale da anticipare l'acquisto e non ordinare articoli in più.

Questi controlli, come dichiarato in precedenza, vengono effettuati solo alla fine dall'algoritmo. Questo implica che la *Tabu search* può potenzialmente avere, in una qualsiasi iterazione, una soluzione non [ammissibile](#) e continuare ad operare con essa. Tuttavia alla fine dell'esecuzione l'algoritmo restituisce una soluzione ottima, che corrisponde alla miglior combinazione per tutti i *range* di copertura di ogni articolo tra data d'ordine e fornitore. L'ottimo dunque non viene influenzato, ma vincolato.

Inoltre se si volessero fare i controlli durante l'esecuzione dell'algoritmo, oltre a rallentarlo notevolmente, si avrebbe un importante aumento di complessità in quanto si dovrebbero salvare le quantità realmente necessarie in un'altra lista e operare altri controlli.

4.4 Codifica

In questa sezione verranno trattate le parti principali e più interessanti riguardo l'attività di codifica e realizzazione dell'algoritmo.

4.4.1 Organizzazione dello sviluppo

Il progetto è stato sviluppato in un ordine preciso. Si è cercato di implementare il tutto utilizzando un approccio incrementale, in modo tale da avere sempre un prodotto funzionante.

In particolare, lo sviluppo è stato svolto nel seguente ordine:

- * creazione della *Windows form*
- * lettura dei dati dal *database*
- * *pre-processing* dei dati
- * costruzione dell'algoritmo *greedy*
- * costruzione di altre scelte *greedy*
- * costruzione della *Tabu search*
- * lettura dei vincoli
- * costruzione dei vari metodi risolutivi per il soddisfacimento dei vincoli

Quest'ordine ha permesso di concentrarsi su obiettivi specifici, facilitandone dunque l'implementazione sia a livello logico che a livello pratico.

Al raggiungimento di ogni *milestone* si organizzava una brevissima discussione con il *tutor* in cui veniva verificata la correttezza e la coerenza di quanto sviluppato in relazione agli obiettivi preposti.

4.4.2 Log

Su richiesta di *Ergon Informatica* è stato aggiunto successivamente il requisito *R40-Q-D* riguardante lo svolgimento da parte del programma di attività di scrittura su *file*. Ad ogni avvio della *Tabu search* viene creato un nuovo *file* denominato come "*Log_aaaa_mm_gg_hh_mm_ss*" dove vengono salvati dati ad ogni iterazione, quali:

- * l'iterazione corrente
- * cosa è accaduto all'interno dell'iterazione (mossa *tabu*, non migliorante, effettuata)
- * il numero di iterazioni non miglioranti
- * il tempo di esecuzione corrente
- * il valore della funzione di valutazione della migliore soluzione corrente

Il salvataggio di queste informazioni ha l'obiettivo di facilitare il *debug* del codice e la rilevazione dei dati ottenuti dall'algoritmo ad ogni iterazione.

Inoltre questi dati possono essere utilizzati anche in futuro per operare confronti fra esecuzioni della *Tabu search* con tipologie di inizializzazione differenti.

4.4.3 Estensioni del progetto

Alla fine del percorso di *stage* è stato richiesto di elencare eventuali *features* ed estensioni che si possono applicare al progetto.

Nuovo algoritmo

La creazione della *Tabu search* e dell'algoritmo *greedy* non esclude lo sviluppo di un nuovo algoritmo. Infatti, grazie all'utilizzo del *design pattern strategy*, è possibile estendere l'interfaccia *AlgorithmStrategy* e creare il corrispondente oggetto del nuovo algoritmo nella classe *Program*. Chiaramente, affinché la classe sia istanziabile, è necessario implementare tutti i metodi dell'interfaccia.

Si potrebbe valutare, in base alle risorse a disposizione, la creazione di un algoritmo genetico oppure di una combinazione tra la *Tabu search* e un altro algoritmo per provare a mettere i risultati a confronto.

Nuova scelta *greedy*

Le scelte *greedy* esistenti sono utilizzate per andare a creare degli algoritmi che forniscono buone soluzioni. Tuttavia possono esistere delle scelte che non sono state ancora considerate e che portano alla creazione di un algoritmo che genera soluzioni migliori.

Grazie all'utilizzo del *design pattern strategy* è possibile estendere la classe astratta *GreedyAlgorithm*, derivante dall'interfaccia *AlgorithmStrategy*.

Una possibile scelta potrebbe essere quella di selezionare l'articolo più economico del fornitore che ha più richieste d'ordine oppure l'articolo più richiesto del fornitore che ha la minor media delle variazioni dei suoi articoli.

Nuova mossa della *Tabu search*

Le mosse attuali della *Tabu search* sono semplici e rapide, creando una bassa perturbazione della miglior soluzione corrente (o *centro del vicinato*). Tuttavia potrebbe essere necessaria una qualche mossa più complessa in modo tale da visitare più velocemente lo spazio delle soluzioni.

Per esempio si potrebbe pensare di aumentare la molteplicità di applicazione della mossa.

Se precedentemente la mossa veniva applicata a un solo codice articolo, si potrebbe provare ad applicarla a k codici articolo, con k deciso dal programmatore. Rimane comunque fondamentale mantenere la randomicità delle scelte, come spiegato nella sezione §4.3.4.

Multithreading

Questa estensione del progetto andrebbe a considerare la creazione di più istanze della *Tabu search* e di farle eseguire su più *thread*. L'idea si basa sul fatto che ogni *Tabu search* viene inizializzata con una diversa soluzione iniziale in modo tale da diversificare la ricerca e intensificarla durante l'esecuzione dell'algoritmo vero e proprio. Ciò potrebbe essere paragonabile alla popolazione dell'algoritmo genetico dove però, nel caso del *multithreading*, non verrebbe effettuata alcuna ricombinazione, ma ogni soluzione verrebbe trattata singolarmente. Dopo che tutti i *thread* hanno terminato l'esecuzione, si andrà a scegliere la soluzione con la funzione di valutazione minore.

4.5 Tecnologie e strumenti

Di seguito viene esposta una panoramica delle tecnologie e degli strumenti utilizzati. Sono stati tutti imposti da *Ergon Informatica* in quanto sono le tecnologie e gli strumenti da loro impiegati per lo sviluppo *software*.

Per l'azienda le caratteristiche determinanti della scelta di queste tecnologie sono riconducibili al soddisfacimento delle seguenti necessità e aspettative:

- * compatibilità con il sistema *ERGDIS*;
- * facilità di apprendimento;
- * ampia disponibilità della documentazione;
- * gradevolezza dell'interfaccia grafica.

Di seguito vengono presentate le tecnologie utilizzate.

C#

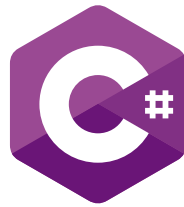


Figura 4.5: Logo *C#*

C# è un linguaggio di programmazione multi-paradigma orientato agli oggetti sviluppato da *Microsoft*. La sua sintassi e struttura derivano da altri linguaggi nati precedentemente, come *C++*, *Java* e *Visual Basic*.

C# è progettato per essere compatibile con le classi e l'ambiente di compilazione del *framework .NET*. Supporta *astrazione*, *ereditarietà* e *polimorfismo* fornendo estensibilità e riusabilità del codice. È stato ufficialmente approvato come *standard* dalla *ECMA*.

L'azienda, sebbene abbia sviluppato la maggior parte dei moduli in *Visual Basic*, sta lentamente traducendo tutto in questo linguaggio.

Visual Studio 2019



Figura 4.6: Logo *Visual Studio 2019*

Visual Studio 2019 è un ambiente di sviluppo integrato (*IDE*) fornito da *Microsoft*. Permette lo sviluppo di *software* per computer, siti web, applicazioni web, servizi web e applicazioni *mobile*. È compatibile con tutte le piattaforme di sviluppo *software Microsoft*, quali *Windows API* e *Windows form*.

Visual Studio 2019 supporta il *refactoring* del codice e *intellisense*, uno strumento per l'autocompletamento del codice. Il *debugger* integrato funziona sia a livello di codice sorgente che a livello di codice macchina. È compatibile anche con i sistemi di supporto per il controllo del codice, come per esempio *Subversion* e *Git*.

Visual Studio 2019 supporta 36 differenti linguaggi di programmazione tra i quali *C#*.

DevExpress



Figura 4.7: Logo *DevExpress*

DevExpress è una compagnia che produce strumenti di sviluppo *software* per *Visual Studio*. In particolare crea estensioni che vengono utilizzate tramite *Visual Studio* per velocizzare la scrittura di applicazioni. Gli strumenti da loro forniti sono molteplici. In particolare hanno sviluppato controlli *.NET* di interfaccia utente (*UI*) che rendono più semplice la creazione di ambienti grafici per applicazioni e portano ad un risultato elegante.

Informix



Figura 4.8: Logo *IBM Informix*

IBM Informix è un prodotto di *IBM*. La base di dati *Informix* è usata in molte applicazioni *OLTP* ad alto tasso di transazione che operano in vari settori tra cui anche quelli della produzione e dei trasporti.

L'*Informix server* supporta il modello relazionale ad oggetti che permette ad *IBM* di offrire estensioni che permettono di effettuare interrogazioni per un dominio specifico e archiviazioni per *set* di dati in maniera rapida ed efficiente. È in grado di supportare sia *SQL* che *NoSQL*.

Git



Figura 4.9: Logo *Git*

Git è uno strumento per il controllo di versione distribuito del codice sorgente delle *repository*. Creato inizialmente per gestire le versioni del *kernel Linux*, al giorno d'oggi è uno dei principali strumenti di versionamento del codice e di collaborazione tra gli sviluppatori.

Capitolo 5

Verifica e validazione

In questo capitolo vengono descritti i processi di verifica e validazione del prodotto, descrivendo i tool utilizzati e le metodologie applicate per valutare il corretto funzionamento e la qualità del prodotto.

5.1 Verifica

In questa sezione vengono esposti gli strumenti e le modalità utilizzate per verificare la correttezza del prodotto durante il progetto.

5.1.1 Modalità

Il progetto di *stage* prevedeva inizialmente lo sviluppo di *test* sia in codice sorgente che documentali.

Il *tutor* successivamente ha consigliato un piano di *test* basato su una semplice esecuzione del programma, variando i suoi parametri, e su un'analisi dei risultati, comparandoli con le rispettive query effettuate tramite l'apposito sistema di interrogazione del *database*.

Questo perché l'attività di prova tramite dati reali comprendeva il lavoro di *debug* permettendo di risparmiare tempo che sarebbe stato altrimenti impiegato nella scrittura di tutti *test*.

Tuttavia, non volendo rinunciare a una parte a mio avviso molto importante dello sviluppo, si è deciso di comune accordo di procedere nel seguente modo: al termine di ogni *milestone* veniva effettuato il *testing* tramite l'esecuzione del programma, venivano corretti tutti i *bug* e venivano poi scritti i *test* per tutte quelle parti di codice con i casi *border line* in modo tale da riuscire a trovare anche piccoli errori non visibili durante l'esecuzione.

Questa modalità è sembrata il giusto compromesso per non sprecare molto tempo nella fase di *testing* e per fornire allo stesso tempo delle dimostrazioni, sotto forma di codice, dei casi limite.

5.1.2 Testing del modulo

Per verificare la correttezza del codice sono stati definiti dei *test* di unità.

Essi sono stati scritti utilizzando il *framework* di *Microsoft MSTest* integrato tramite estensioni in *Visual Studio* e come metodologia di approccio è stato usato il *pattern AAA* (*Arrange, Act, Assert*). Ogni *test* effettua i seguenti passi:

1. **Arrange:** vengono definite e riempite le strutture dati del problema con dei dati creati *ad hoc*;
2. **Act:** viene effettuata la chiamata che agisce sulle strutture create;
3. **Assert:** vengono fatte delle asserzioni, verificando dunque che i valori di *output* siano quelli attesi.

5.2 Validazione

In questa sezione viene descritto il processo di validazione del progetto.

5.2.1 Codice

La validazione del codice è stata effettuata dal *tutor* aziendale, al quale veniva periodicamente illustrato il codice ad alto livello. In ogni fase venivano effettuate delle critiche costruttive e dati dei consigli da applicare nella fase successiva. Questo tipo di validazione mirava a verificare che il codice avesse certe caratteristiche in termini di leggibilità e correttezza.

5.2.2 Requisiti

Attraverso l'attività di *testing* è stato validato anche il raggiungimento dei requisiti rilevati durante l'analisi iniziale. Tutti i requisiti obbligatori sono stati raggiunti e rappresentavano la condizione necessaria per superare l'attività di *stage*.

Alcuni requisiti desiderabili e facoltativi non sono stati soddisfatti a causa della mancanza di tempo. Tuttavia si è preferito concentrarsi solo su alcuni di essi e soddisfarli pienamente piuttosto che eseguirli tutti in modo non esaustivo.

Nelle tabelle 5.1, 5.2, 5.3 e 5.4 esposte di seguito viene presentato il risultato della validazione.

Tabella 5.1: Tabella della validazione dei requisiti funzionali

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

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

R33-F-O	L'utente deve poter ordinare la lista rispetto alla data iniziale di copertura	UC6.4	Soddisfatto
R34-F-O	L'utente deve poter ordinare la lista rispetto alla data finale di copertura	UC6.5	Soddisfatto
R35-F-O	L'utente deve poter ordinare la lista rispetto alla quantità ordinata	UC6.6	Soddisfatto
R36-F-O	L'utente deve poter ordinare la lista rispetto al prezzo parziale	UC6.7	Soddisfatto

Tabella 5.2: Tabella della validazione dei requisiti qualitativi

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

Tabella 5.3: Tabella della validazione dei requisiti di performance

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

Tabella 5.4: Tabella della validazione dei requisiti di vincolo

Requisito	Descrizione	Use Case	Risultato
R46-V-O	La <i>form</i> deve essere eseguita sull'ambiente di esecuzione <i>.NET Framework</i>	-	Soddisfatto
R47-V-O	La <i>form</i> e l'algoritmo devono essere codificate in <i>C#</i>	-	Soddisfatto
R48-V-O	La versione utilizzata di <i>C#</i> deve essere 7.3	-	Soddisfatto
R49-V-O	La versione utilizzata di <i>.NET Framework</i> deve essere 4.8	-	Soddisfatto
R50-V-O	L'algoritmo finale deve fornire una soluzione ammissibile	-	Soddisfatto

5.2.3 Risultati dei test

Di seguito vengono illustrati alcuni risultati riguardanti i *test* effettuati sulla *Tabu search*, cercando di variare:

1. il numero di iterazioni massime
2. la scelta della risoluzione dei minimi

Dato che il *database* fornitomi dall'azienda conteneva delle incongruenze fra i campi dati delle rispettive tabelle, si è deciso di raccogliere le rilevazioni su un circoscritto numero di articoli. Questi infatti sono stati prima verificati tramite l'utilizzo del sistema di query integrato in *ERGDIS* in modo tale da non ritrovarsi risultati errati a causa di circostanze non dipendenti dall'algoritmo.

Variazione numero massimo di iterazioni

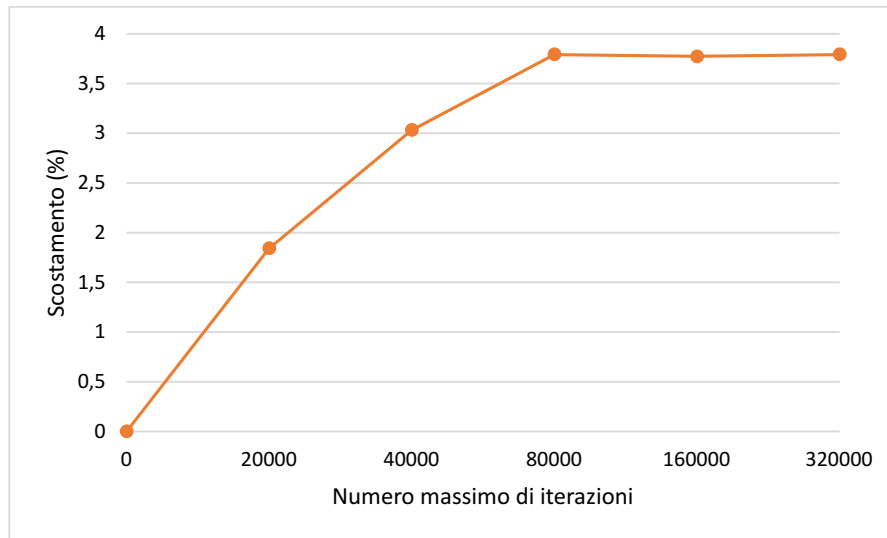
Dati in *input*:

- * tipo di inizializzazione: *HighestArtRequestByArt*
- * tipo di risoluzione dei minimi: *Min/Max*
- * numero massimo di iterazioni non miglioranti: 2500
- * grandezza della *Tabu list*: 200
- * tempo di esecuzione massimo: 10 minuti (da requisito *R45-P-O*)

Tabella 5.5: Test - variazione del numero massimo di iterazioni

MaxIt	PreTot (€)	PostTot (€)	Scostamento (%)
20.000	24.876,15	24.418,43	1,84
40.000		24.122,40	3,03
80.000		23.933,34	3,79
160.000		23.938,32	3,77
320.000		23.933,34	3,79

Figura 5.1: Grafico scostamento/iterazioni



Si osservi la tabella 5.5.

Lo scostamento percentuale dei totali inizialmente aumenta, per poi rimanere in stallo raggiunto un certo numero di iterazioni. Questo accade poichè, molto probabilmente, si è raggiunto un buon minimo locale e la probabilità di trovare una mossa migliorante diminuisce vertiginosamente all'aumentare del numero di iterazioni effettuate.

Confronto fra le scelte di risoluzione dei minimi

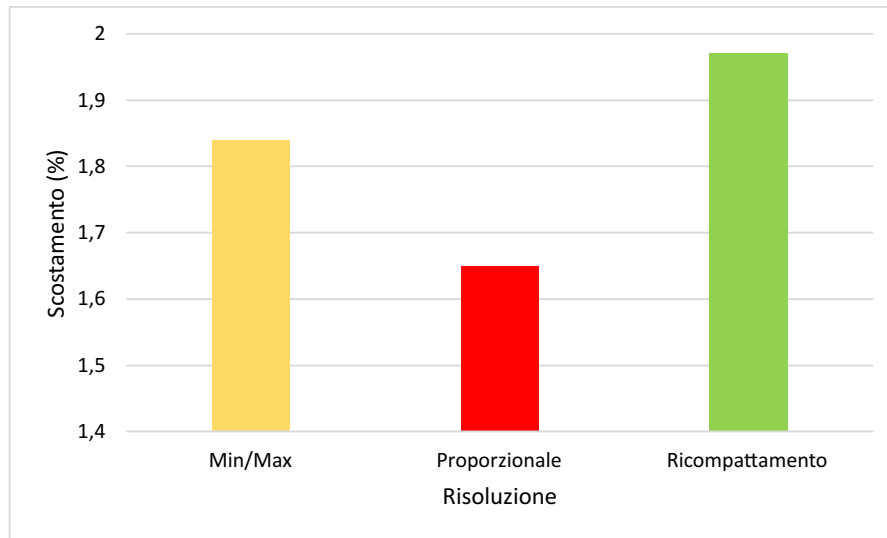
Dati in *input*:

- * tipo di inizializzazione: *HighestArtRequestByArt*
- * numero massimo di iterazioni: 20000
- * numero massimo di iterazioni non miglioranti: 2500
- * grandezza della *Tabu list*: 200
- * tempo di esecuzione massimo: 10 minuti (da requisito [R45-P-O](#))

Tabella 5.6: *Test* - scelta di risoluzione dei minimi

Risoluzione	PreTot (€)	PostTot (€)	Scostamento (%)
<i>Min/Max</i>	24.876,15	24.418,43	1,84
Proporzionale		24.465,69	1,65
Ricompattamento		24.386,09	1,97

Figura 5.2: Grafico scostamento/risoluzione



Si osservi la tabella [5.6](#).

La scelta peggiore per la risoluzione dei minimi è l'utilizzo del metodo proporzionale. Questo risultato non dovrebbe essere una sorpresa. Infatti, nel controllo del minimo relativo al singolo ordine, si ha una più alta probabilità di aumentare di una unità la quantità di un ordine con prezzo unitario che non corrisponde al minimo.

Il ricompattamento invece provoca lo scostamento maggiore di tutti e questo è abbastanza ovvio in quanto è una mossa utile per evitare di comprare articoli in eccesso, andando così a ridurre le scorte di magazzino (e conseguentemente anche il prezzo totale speso).

Capitolo 6

Conclusioni

In questo ultimo capitolo viene effettuata una analisi retrospettiva sullo stage focalizzandosi sul raggiungimento degli obiettivi, sulle conoscenze acquisite e sulla valutazione personale del percorso.

6.1 Prodotto finale

Come descritto nei capitoli precedenti, il prodotto finale consiste in una *Windows form* che ottimizza l'insieme degli ordini che vengono effettuati per soddisfare i fabbisogni di un dato periodo.

Dopo aver inserito tutti gli *input* necessari, viene determinata una soluzione iniziale tramite un algoritmo *greedy* e poi, attraverso la *Tabu search*, si crea una nuova soluzione che viene visualizzata in una lista filtrabile contenente tutte le informazioni di ogni ordine.

6.2 Raggiungimento degli obiettivi

Come descritto nella sezione §5.2.2, il prodotto soddisfa la maggior parte delle necessità descritte nell'analisi dei requisiti. In particolare sono stati realizzati tutti i requisiti funzionali, la maggior parte di quelli qualitativi, tutti quelli di performance e di vincolo.

Nella tabella 6.1 viene fornita una chiara visuale del soddisfacimento dei requisiti.

Tabella 6.1: Riepilogo della validazione dei requisiti

Tipo	Obbligatori	Desiderabili	Facoltativi
Soddisfatti	45	2	0
Non Soddisfatti	0	2	1

6.3 Conoscenze acquisite

Per la realizzazione di questo progetto sono state fondamentali molte nozioni apprese durante il corso di studi, dai linguaggi ai paradigmi.

Lo *stage* ha permesso però anche di incrementare notevolmente il mio bagaglio sia tecnico che personale. Vengono ora elencate le principali conoscenze acquisite.

Informix, C# e MSTest

Per lo sviluppo del progetto mi sono servito principalmente di queste tecnologie:

- * *Informix* per *database*
- * *C#* come linguaggio di programmazione
- * *MSTest* come *framework* di *test*

Non avendole mai affrontate né all'università né in privato ho dovuto cominciare praticamente da zero, ma poichè struttura, regole e utilizzo sono simili a tecnologie conosciute, come ad esempio *MySQL* per *Informix*, *C++* e *Java* per *C#*, e *JUnit* per *MSTest*, l'apprendimento è stato molto semplificato. Ciò mi ha permesso di acquisire le conoscenze necessarie in tempi rapidissimi e grazie anche all'utilizzo di un approccio conosciuto come *learning by doing*.

Analisi e modellazione del problema

Analizzare e modellare un problema è una delle competenze che si devono apprendere in un corso di laurea in informatica. Tuttavia i problemi con cui si ha a che fare nel mondo accademico sono molto spesso *standard* e risolvibili tramite tecniche ben conosciute.

Questo progetto mi ha posto davanti per la prima volta un problema non banale da risolvere da solo. Infatti uno studio preliminare forse poco accurato mi ha portato a effettuare per buona parte dello *stage* molte domande al *tutor*. È chiaro dunque come sia fondamentale capire il problema che si ha di fronte nella sua completa interezza prima di iniziare la progettazione e lo sviluppo.

Gestione delle risorse

Il mondo universitario pone lo studente a organizzare le sue risorse temporali disponibili per preparare al meglio gli esami. Tutto ciò non ha nulla a che vedere con la gestione delle risorse in ambito aziendale. L'organizzazione diventa fondamentale e la schedulazione degli eventi e degli obiettivi diventa imprescindibile. Lo svolgimento dello *stage* mi ha permesso dunque di crescere molto su questo aspetto che avevo già rafforzato nel corso di *Ingegneria del software*.

6.4 Valutazione complessiva

Lo *stage* presso *Ergon Informatica* lo valuto in maniera molto positiva.

I due mesi trascorsi in azienda sono risultati molto leggeri e piacevoli soprattutto grazie ad un ambiente di lavoro molto motivante e, a tratti, anche divertente.

Grazie ad un ottimo rapporto con il *tutor* aziendale, non ho mai dovuto affrontare difficoltà bloccanti in autonomia. Gianluca, anche durante il suo periodo di vacanza, è sempre stato disponibile ad ascoltare dubbi e perplessità per fornirmi indicazioni utili che mi guidassero verso la via corretta e mi permettessero di continuare il lavoro.

Un altro aspetto che ho molto apprezzato è stato l'interesse di alcuni colleghi verso il progetto che stavo realizzando, il che mi faceva sentire parte integrante dell'azienda.

Per quanto concerne la parte di realizzazione del progetto, tutto è proseguito secondo quanto programmato.

Personalmente ho trovato più difficile la parte di inquadramento del problema e di scelta della funzione di valutazione. Quest'ultima è stata particolarmente complessa perchè non esisteva una qualche procedura che descrivesse come arrivare a definire la funzione di valutazione, ma bisognava effettuare delle prove empiriche e aggiustare la funzione tentativo dopo tentativo. Ho avuto modo di affrontare tematiche nuove legate alla risoluzione di problemi relativi al mondo reale e questo mi ha permesso di rimanere costantemente motivato.

Concludendo, questo progetto di *stage* è stato per me molto soddisfacente per modalità di svolgimento, per il prodotto finale, per le conoscenze acquisite e per le persone conosciute.

Glossario

.NET Framework Ambiente di esecuzione runtime della piattaforma tecnologica *.NET* in cui vengono gestite le applicazioni destinate allo stesso *.NET Framework*. [2](#)

Arrange-Act-Assert *pattern* per scrivere *test* di unità che abbiano una struttura uniforme. Questo li rende anche facilmente leggibili e comprensibili. [47](#)

Ammissibile Soluzione che rispetta tutti i vincoli del problema. [8](#), [31](#), [33](#), [41](#), [51](#)

API (*Application Programming Interface*) intermediario *software* che permette a due applicazioni non correlate di comunicare tra loro. [44](#)

Astrazione Applicazione del metodo logico di astrazione nella strutturazione della descrizione dei sistemi informatici complessi, per facilitarne la progettazione e manutenzione o la stessa comprensione. La pratica consiste nel presentare il sistema, ad esempio un pezzo di codice sorgente o uno scambio di trasmissioni di dati, in maniera ridotta ai soli dettagli considerati essenziali all'interesse specifico, ad esempio raggruppando il codice in una funzione o formalizzando un protocollo di comunicazione. [44](#)

Business Intelligence Con la locuzione *business intelligence* (*BI*) ci si può solitamente riferire a: un insieme di processi aziendali per raccogliere dati ed analizzare informazioni strategiche, la tecnologia utilizzata per realizzare questi processi oppure le informazioni ottenute come risultato di questi processi. [2](#)

C++ Linguaggio di programmazione general purpose sviluppato come evoluzione del linguaggio C inserendo la programmazione orientata agli oggetti. [4](#), [44](#), [56](#)

Database Archivio di dati strutturato in modo da razionalizzare la gestione e l'aggiornamento delle informazioni e da permettere lo svolgimento di ricerche complesse. [4](#), [33](#), [42](#), [47](#), [51](#), [55](#)

Debug Termine che indica l'attività che consiste nell'individuazione e correzione da parte del programmatore di uno o più errori (*bug*) rilevati nel *software*, direttamente in fase di programmazione oppure a seguito della fase di *testing* o dell'utilizzo finale del programma stesso. [42](#), [47](#)

Debugger Programma/*software* specificatamente progettato per l'analisi e l'eliminazione dei *bug* (*debugging*), ovvero errori di programmazione interni al codice di altri programmi. [44](#)

DevExpress *Framework* utile per lo sviluppo di applicazioni desktop. [2](#), [4](#), [45](#)

Ereditarietà Uno dei Principio fondamentale della programmazione ad oggetti. In generale, essa rappresenta un meccanismo che consente di creare nuovi oggetti che siano basati su altri già definiti. Si definisce oggetto figlio quello che eredita tutte o parte delle proprietà e dei metodi definiti nell'oggetto padre. [44](#)

ERGDIS *Software* proprietario dell'azienda *Ergon Informatica S.R.L.* [1](#), [2](#), [4](#), [44](#), [51](#)

- ERP** (*Enterprise Resource Planning*) 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](#)
- IDE** (*Integrated Development Environment*) *Software* che aiuta gli sviluppatori nella fase di programmazione con vari strumenti di analisi del codice e integrazione di altre componenti. [4](#), [44](#)
- IntelliSense** Forma di completamento automatico resa popolare da Visual Studio Serve inoltre come documentazione per i nomi delle variabili, delle funzioni e dei metodi usando metadati. L'uso dell'IntelliSense è un metodo conveniente per visualizzare la descrizione delle funzioni, in particolar modo la lista dei loro parametri. Questa tecnologia riesce a velocizzare lo sviluppo del *software* riducendo la quantità di *input* attraverso la tastiera. [44](#)
- Java** linguaggio di programmazione *general-purpose* ad livello, basato su classi e orientato alla programmazione ad oggetti. Designato per avere il minor numero di dipendenze di implementazione possibile. [4](#), [44](#), [56](#)
- JUnit** *Framework* di *unit testing* per il linguaggio di programmazione *Java*. [56](#)
- LINQ** (*Language-Integrated Query*) Insieme di tecnologie basate sull'integrazione delle funzionalità di query direttamente nel linguaggio C#. [4](#)
- Log** *File* su cui sono registrati eventi caratteristici dell'applicazione e che fungono in certi casi da vero e proprio protocollo di entrata e di uscita. Ad esempio in programmazione il *file* di *log* evidenzia il tipo di errore e il punto nel codice da parte dell'*IDE*. [30](#), [50](#)
- Milestone** Consiste in importanti traguardi intermedi nello svolgimento di un progetto. [42](#), [47](#)
- NoSQL** Tipo di *database* realizzati per modelli di dati specifici con schemi flessibili. Sono molto utilizzati per la facilità di sviluppo, le funzionalità che offrono e la scalabilità delle prestazioni. [45](#)
- OLTP** (*Online transaction processing*) è un insieme di tecniche *software* utilizzate per la gestione di applicazioni orientate alle transazioni. [45](#)
- Polimorfismo** il termine usato in senso generico per riferirsi a espressioni che possono rappresentare valori di diversi tipi (dette espressioni polimorfiche). In un linguaggio non tipizzato, tutte le espressioni sono intrinsecamente polimorfiche. Nel contesto della programmazione orientata agli oggetti, si riferisce al fatto che una espressione il cui tipo sia descritto da una classe A può assumere valori di un qualunque tipo descritto da una classe B sottoclasse di A (polimorfismo per inclusione). [44](#)
- Pre-processing** Consiste nella manipolazione o nell'eliminazione dei dati prima che vengano utilizzati al fine di garantire o migliorare le prestazioni. [33](#), [42](#)
- Refactoring** Tecnica per modificare la struttura interna di porzioni di codice senza modificarne il comportamento esterno, applicata per migliorare alcune caratteristiche non funzionali del *software*. [44](#)
- Ricerca locale** Algoritmo che si basa sull'idea di migliorare una soluzione iniziale esplorandone un intorno opportunamente definito. Se l'ottimizzazione dell'intorno produce una soluzione migliorante il procedimento viene ripetuto considerando come soluzione corrente la soluzione appena determinata. L'algoritmo termina se non è più possibile trovare soluzioni miglioranti, quindi al raggiungimento di un ottimo locale, oppure al raggiungimento di un numero determinato di iterazioni o di un tempo massimo di esecuzione. [9](#)

Ricerca operativa Scienza che fornisce strumenti matematici di supporto alle attività decisionali in cui occorre gestire e coordinare attività e risorse limitate. [4](#), [7](#)

Server Consolidation 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](#)

SQL (Structured Query Language) Linguaggio standardizzato per *database* basati sul modello relazionale. [45](#)

Stakeholders Coloro che a vario titolo hanno influenza su un prodotto o su un progetto. [2](#)

UML (*Unified Model Language*) *Standard* utilizzato nell'ingegneria del software per descrivere un sistema informatico, consentendo ai vari ruoli (sviluppatore, tester, analista,ecc.) di comunicare tramite lo stesso linguaggio. Si avvale di diversi tipi di diagrammi, statici e dinamici, per fornire diverse viste di uno stesso sistema. [13](#)

Visual Basic Linguaggio di programmazione *event driven* creato da *Microsoft*. [44](#)

Windows form Applicazione basata su eventi supportata da *.NET Framework* di *Microsoft*. [3](#), [34](#), [42](#), [44](#), [55](#)

Bibliografia

Riferimenti bibliografici

Gamma, E. *Design Patterns*. Pearson Education, Addison-Wesley, 1994.

Sommerville, Ian. *Software Engineering, 10th edition*. Pearson Education, Addison-Wesley, 2018.

Siti web consultati

A New MRP Optimization Algorithm Based on Multi-Objective Genetic Evolution. URL: <https://www.atlantispress.com/proceedings/iske2007/1351>.

Azienda Ergon Informatica. URL: <https://www.ergon.it/>.

Definizione di alcuni termini del glossario. URL: <https://www.wikipedia.org/>.

Dispense UML e design pattern. URL: <https://www.math.unipd.it/~rcardin/swea/2021/>.

Documentazione C#. URL: <https://docs.microsoft.com/en-us/dotnet/csharp/>.

Documentazione DevExpress. URL: <https://docs.devexpress.com/>.

Documentazione Entity Framework. URL: <https://docs.microsoft.com/en-us/ef/>.

Documentazione Informix. URL: <https://www.ibm.com/docs/en/informix-servers/14.10>.

Documentazione LINQ. URL: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/>.

Documentazione MSTest. URL: <https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-mstest>.

Documentazione Visual Studio. URL: <https://docs.microsoft.com/en-us/visualstudio/windows/?view=vs-2019>.

Introduction to Genetic Algorithms. URL: <https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3>.

Local Optimization Versus Global Optimization. URL: <https://machinelearningmastery.com/local-optimization-versus-global-optimization/#:~:text=Local%20optimization%20involves%20finding%20the,problems%20that%20contain%20local%20optima..>

Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. URL: https://www.iiia.csic.es/~christian.blum/downloads/blum_rol_i_2003.pdf.

Metodi euristici di ottimizzazione combinatoria. URL: <https://www.math.unipd.it/~luigi/courses/metmodoc1011/m08.meta.pdf>.

Optimization Techniques — Tabu Search. URL: <https://towardsdatascience.com/optimization-techniques-tabu-search-36f197ef8e25>.

Tabu search and genetic algorithms: a comparative study between pure and hybrid agents. URL: https://abepro.org.br/biblioteca/enegep2006_ti450301_7406.pdf.