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

Anno Accademico 2021-2022

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

— *Isaac Asimov*

Ringraziamenti

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

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

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

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

Padova, Settembre 2022

Filippo Brugnolaro

Indice

Elenco delle figure

Elenco delle tabelle

Lista degli algoritmi

Capitolo 1

Introduzione

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

1.1 L'azienda

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

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

Il logo dell'azienda è illustrato in Figura 1.1.

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

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

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

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

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

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

1.2 L'idea

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

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

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

1.3 Descrizione dello stage

1.3.1 Introduzione

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

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

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

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

Tabella 1.1: Esempio - Fabbisogni

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

Tabella 1.2: Esempio - Listino Prezzi

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

Tabella 1.3: Esempio - Calendario spedizioni

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

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

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

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

1.3.2 Obiettivi

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

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

1.3.3 Analisi preventiva dei rischi

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

1. **Comprensione e confronto degli algoritmi**

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

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

2. **Tecnologie e ambiente di sviluppo**

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

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

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

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

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

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

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

1.4 Organizzazione del testo

Di seguito viene illustrata l'organizzazione dei capitoli successivi:

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

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

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

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

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

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

* gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario alla fine del presente documento;
* i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Studio di fattibilità

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

2.1 Introduzione allo studio

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

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

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

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

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

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

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

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

2.2 Soluzioni proposte

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

2.2.1 Algoritmo Greedy

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

Pseudocodice string replacement - Algoritmo Greedy

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

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

Aspetti positivi

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

Aspetti negativi

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

2.2.2 Tabu Search

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

Pseudocodice string replacement - Tabu Search

Input: $stringa\_corretta$, $stringa\_errata$, $capienza\_tabu\_list$, $max\_iterazioni$
Output: $funzione\_obiettivo$

1: **procedure** MY_TABU_SEARCH($stringa\_corretta$, $stringa\_errata$, $capienza\_tabu\_list$, $max\_iterazioni$)
2: $sol_{curr} \leftarrow stringa\_errata$
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 appartenza della mossa alla $tabu\_list$, non fa calcolare e conseguentemente fare il confronto con $funzione\_obiettivo_{best}$. Inoltre se la $funzione\_obiettivo_{curr}$ non è migliorante, l'iterazione viene comunque contata.
Per quanto riguarda la condizione di **break**, essa viene messa poichè, avendo trovato una soluzione migliorante, non è più necessario esplorare il vicinato corrente ed è dunque necessario crearne uno nuovo a partire dalla nuova soluzione.

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

Pseudocodice string replacement - Algoritmo Genetico

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

procedure MY_GENETIC_ALGORITHM($stringa\_corretta$, $stringa\_errata$, $crossover\_rate$, $mutation\_rate$, $max\_iterazioni$)
    $arr_{str\_corr} \leftarrow codifica(stringa\_corretta)$
    $lista\_pop \leftarrow inizializza\_pop()$
    $set\_fitness()$
    $fitness_{best} \leftarrow most\_fitness\_val(arr_{str\_corr})$
    $sol_{best} \leftarrow most\_fitness\_sol(arr_{str\_corr})$
    $num_{crossover} \leftarrow calculate\_num\_crossover(crossover\_rate, lista\_pop)$
    $conta \leftarrow 0$

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

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

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

Aspetti positivi

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

Aspetti negativi

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

2.3 Conclusioni dello studio

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

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

Tipologia	**Efficacia (%)**	**Efficienza (ms)**	**Tempo di realizzazione (h)**
Algoritmo Greedy	100	0,45	0,5
Tabu Search	98	1289,76	2,5
Algortimo 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 Introduzione).
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 sposta 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.

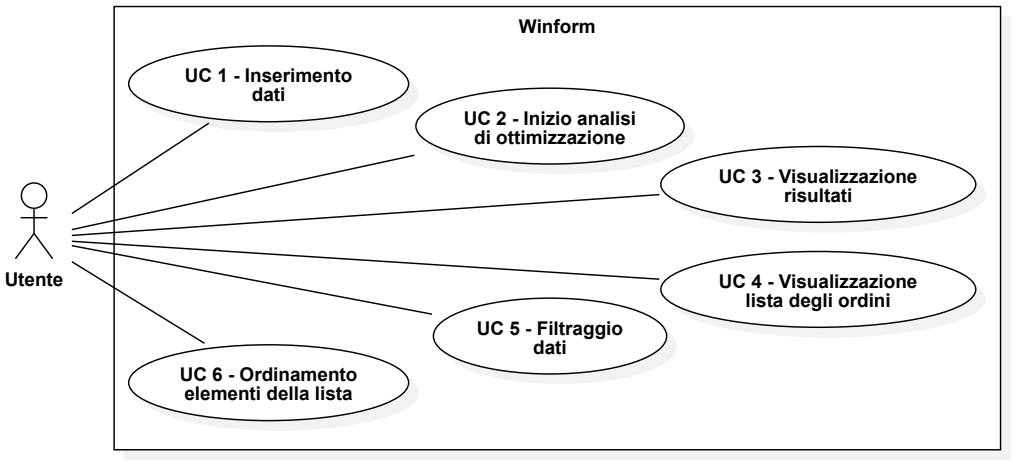

Figura 3.1: Use case - sistema principale

UC 1 - Inserimento dati

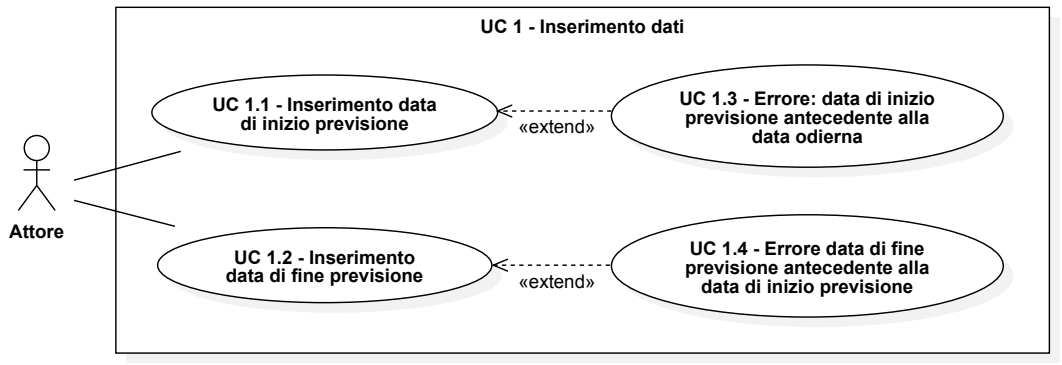

Figura 3.2: UC1 - Inserimento dati

* **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

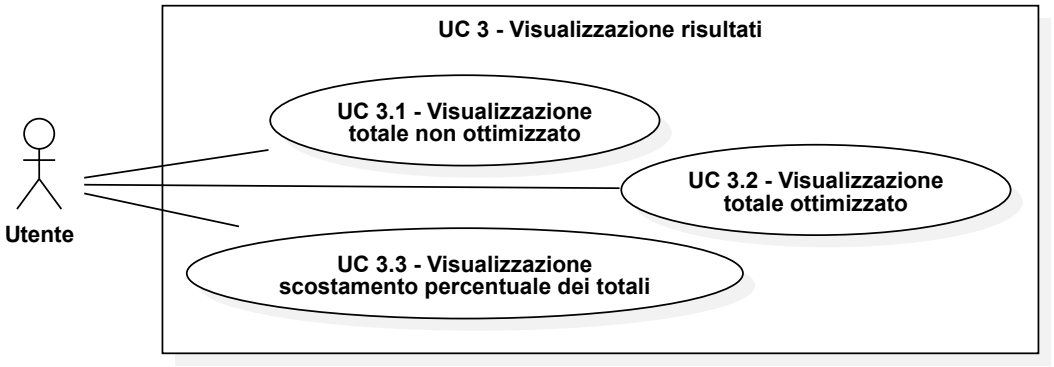

Figura 3.3: UC3 - Visualizzazione risultati

* **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

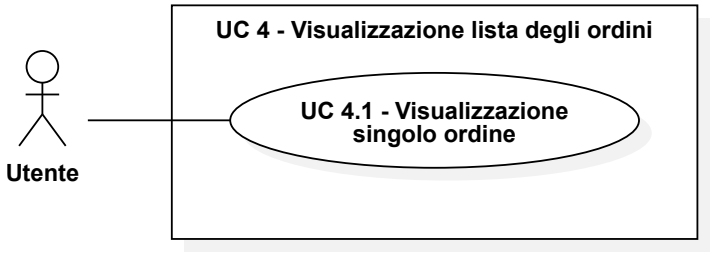

Figura 3.4: UC4 - Visualizzazione lista degli ordini

* **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

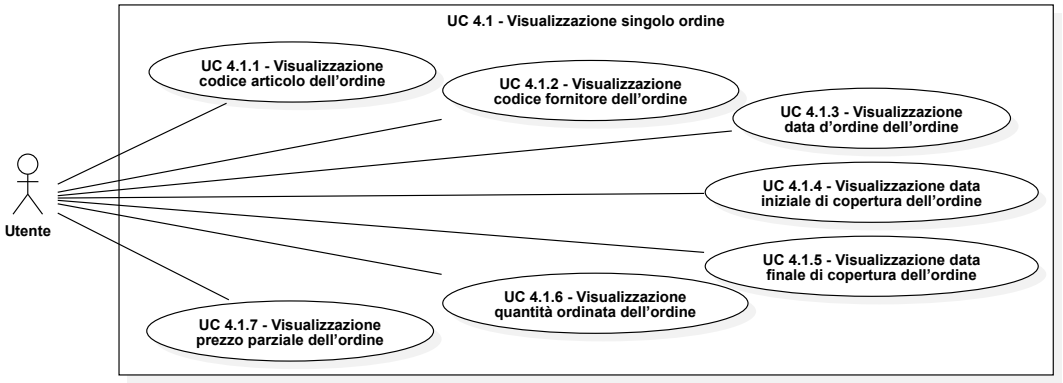

Figura 3.5: UC4.1 - Visualizzazione singolo ordine

* **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

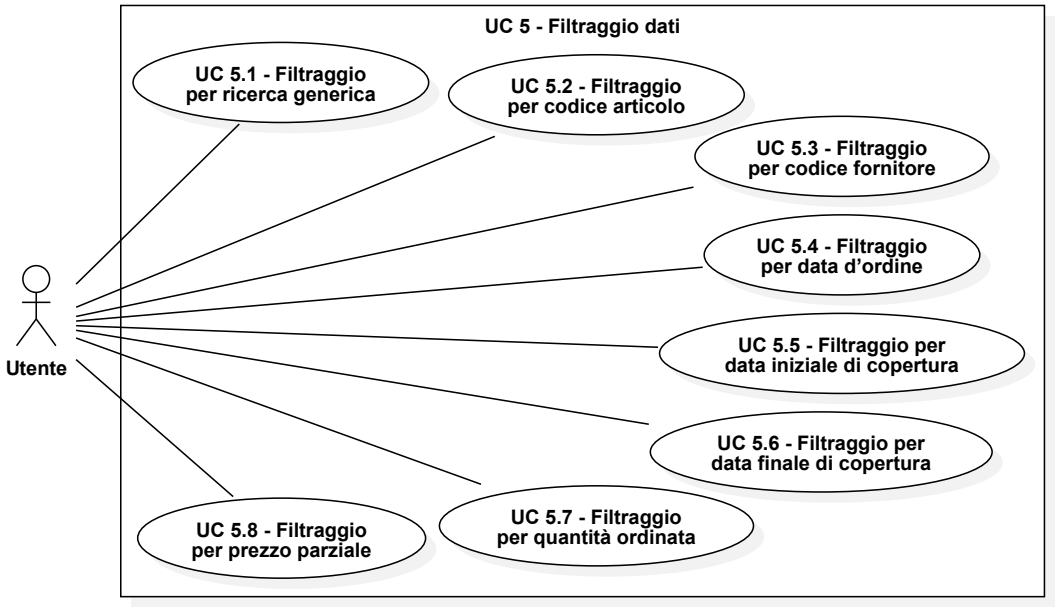

Figura 3.6: UC 5 - Filtraggio dati

* **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

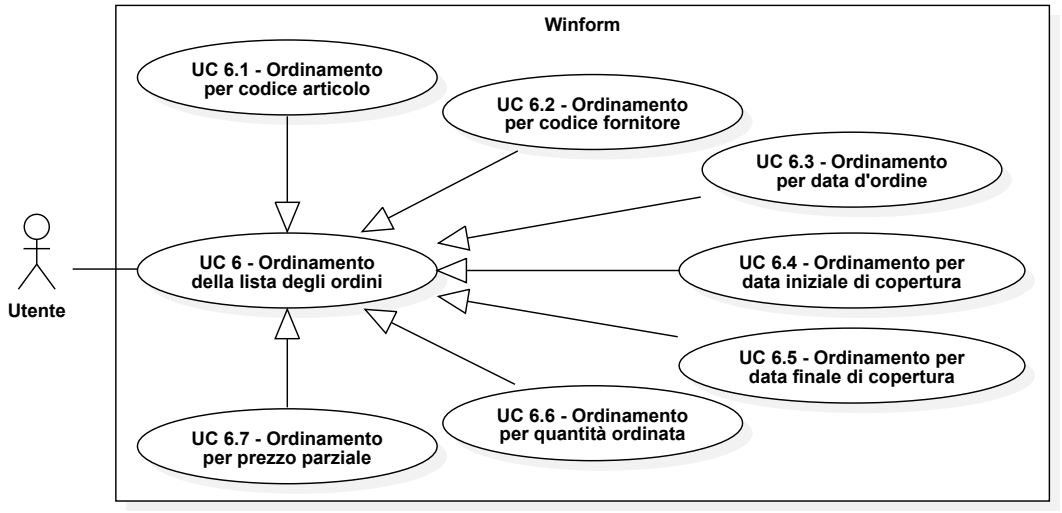

Figura 3.7: UC 6 - Ordinamento della lista degli ordini

* **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:

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

In particolare il tipo può assumere 4 valori, quali:

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

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

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

Nelle tabelle 3.1, 3.2, 3.3 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 log di chiamata per manutenzioni future	-
R41-Q-D	L'algoritmo di ottimizzazione deve essere estensibile	-
R42-Q-D	L'algoritmo utilizza differenti tecniche di ottimizzazione	-
R43-Q-F	L'algoritmo utilizza il multithreading per cercare più soluzioni ammissibili	-

Tabella 3.3: Tabella del tracciamento dei requisiti di performance

Requisito	Descrizione	Use Case
R44-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
R45-V-O	La *form* deve essere eseguita sull'ambiente di esecuzione *.NET Framework*	-
R46-V-O	La *form* e l'algoritmo devono essere codificate in *C#*	-
R47-V-O	La versione utilizzata di *C#* deve essere 7.3	-
R48-V-O	La versione utilizzata di *.NET Framework* deve essere 4.8	-
R49-V-O	L'algoritmo finale deve fornire una soluzione ammissibile	-

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.

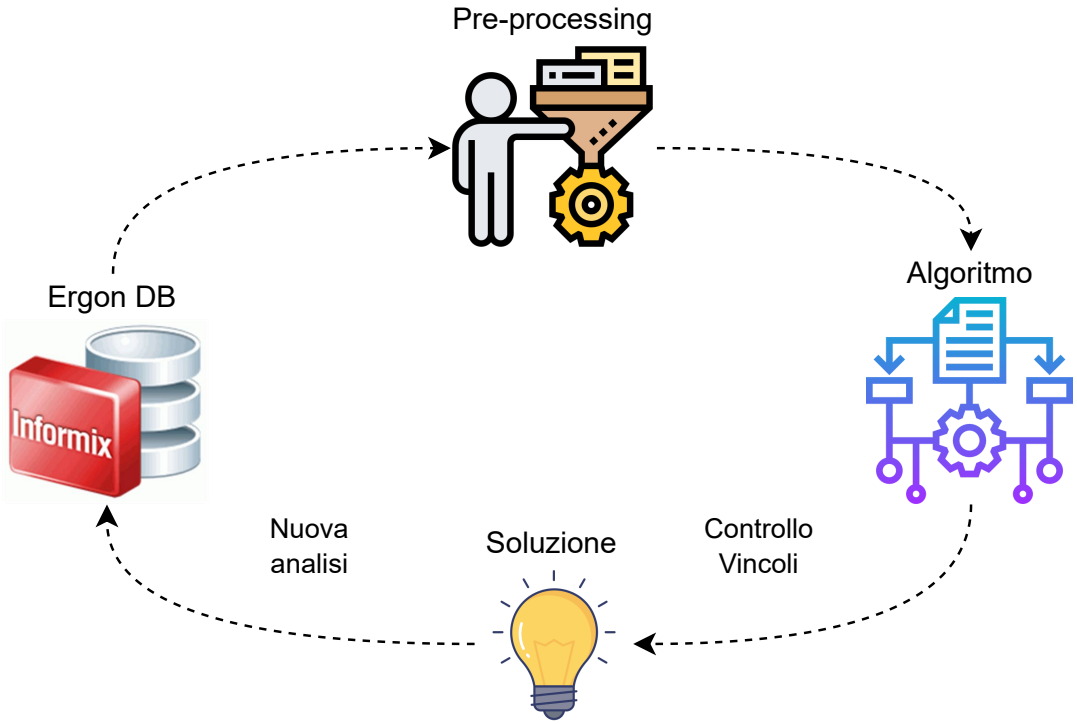

Figura 4.1: Architettura generale

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 ed eseguire nuovamente 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.

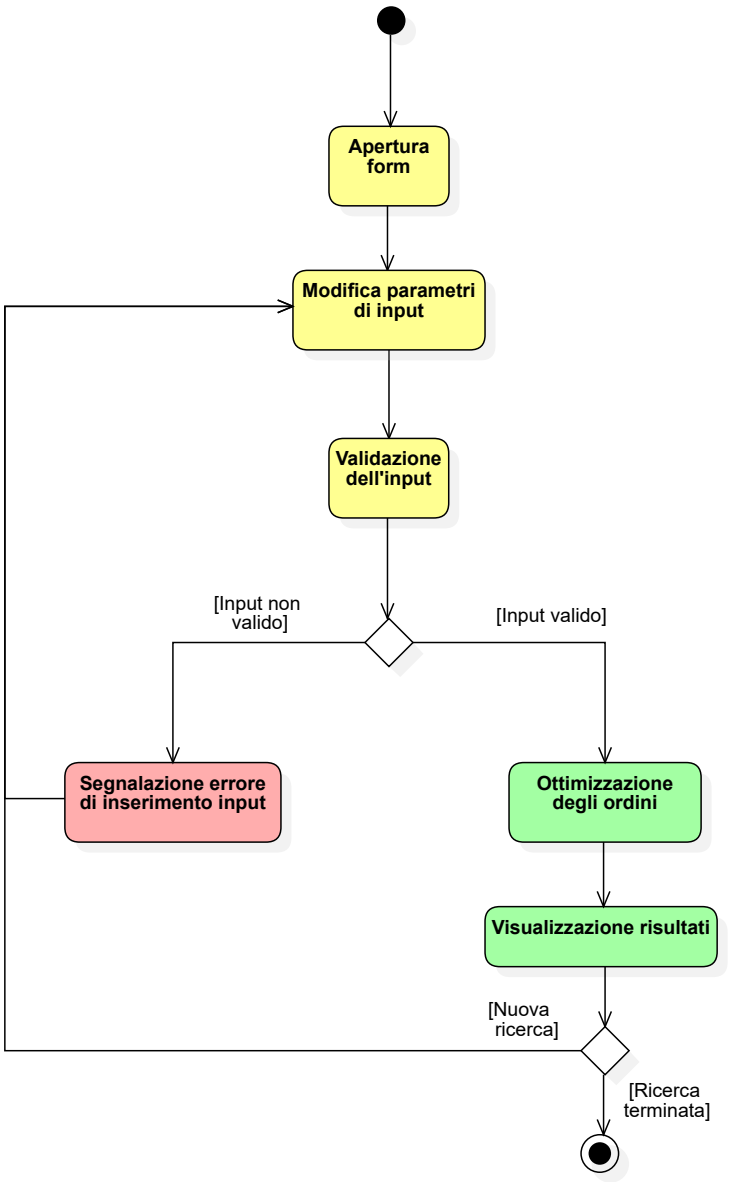

Figura 4.2: Diagramma di attività della WinForm

L'utente, all'apertura della WinForm, 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.

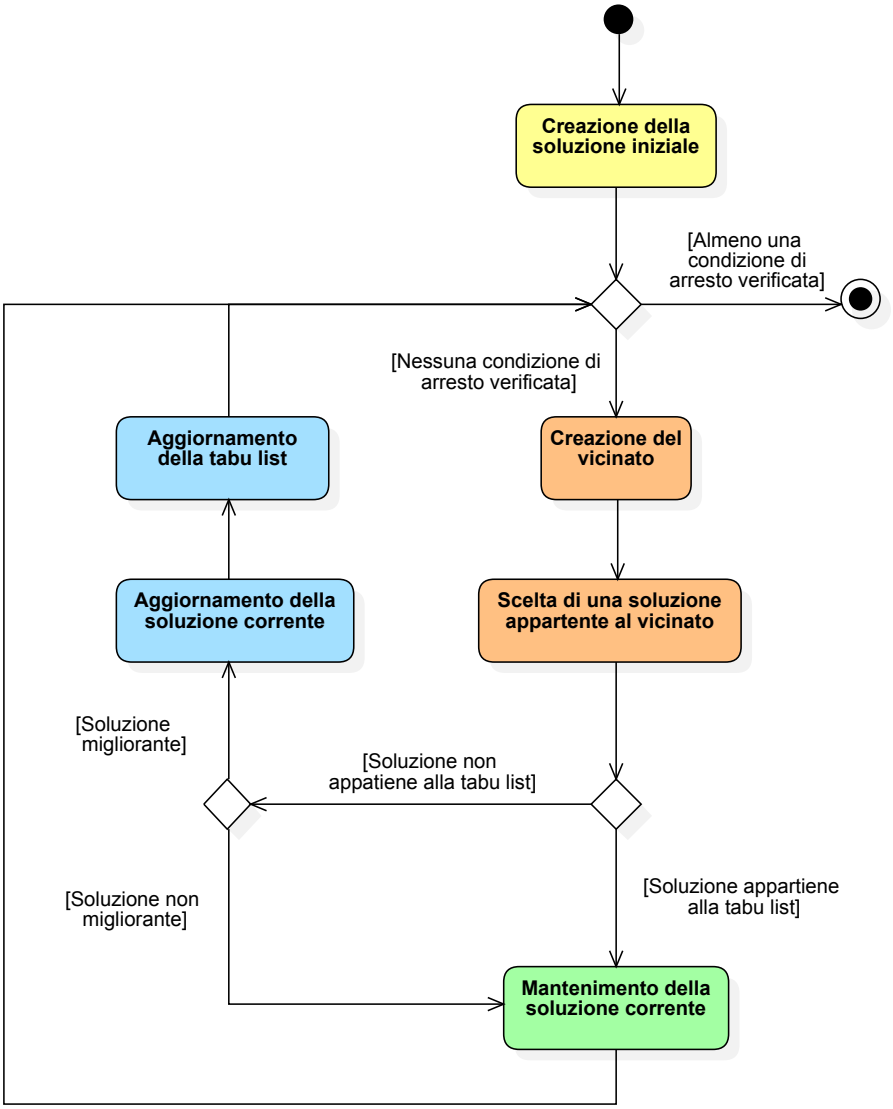

Figura 4.3: Diagramma di attività della tabu search

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.

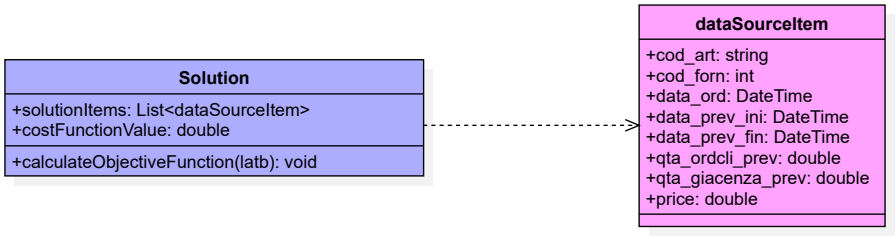

Figura 4.4: Rappresentazione della soluzione per la tabu search

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.
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: $l \leftarrow lista\_vuota$
3: $l_{tmp} \leftarrow lista\_vuota$
4: $l_{cod\_art} \leftarrow genera\_lista\_decisione(lista\_articoli)$
5: $n_{cod\_art} \leftarrow l_{cod\_art}.Length$
6: **while** $count < n_{cod\_art}$ **do**
7: $cod\_art \leftarrow decisione(l_{cod\_art})$
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
* dato un fornitore, non esiste alcuna spedizione associata ad un prezzo

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 ogni range di copertura di ogni articolo viene preso il record con il fornitore che offre il fabbisogno al prezzo più vantaggioso.

É 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 ogni range di copertura di ogni articolo sia necessario scegliere la miglior combinazione tra data d'ordine e fornitore.
Data la sua alta complessità non è dunque possibile visitare l'intero vicinato, motivo per il quale si è deciso di procedere randomizzando le scelte. In particolare ogni mossa viene selezionata in maniera casuale cosicché tutte le scelte abbiano la stessa probabilità di essere selezionate.
Per lo stesso motivo è stato applicato lo stesso principio anche per la scelta dell'articolo su cui applicare la mossa.
É 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, viene utilizzato 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 lo stesso numero, cosa assolutamente 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 alla creazione dell'oggetto *TabuSearch*.
Permette di evitare dunque cicli infiniti in corrispondenza di minimi locali. Questo accade perché tramite le funzione $generate\_move\_string()$ ogni mossa viene prima codificata, poi viene effettuata una ricerca nella tabu list, impedendo potenzialmente all'algoritmo di riprovarla, e infine viene aggiunta nella lista sia nel caso in cui venga effettuata che nel caso in cui non venga effettuata 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 la bontà di 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 il prezzo totale diminuisce a parità di articoli ordinati. 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, 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:

$$f = \frac{PT \cdot (1-R)}{e^R}$$

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

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

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

É evidente dalla tabella 4.1 come la funzione f sia migliorativa se il prezzo totale degli ordini è minore, ma non lo sia nel caso di aumento del numero di articoli ordinati. 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 minore
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 inizia a decrescere. Questo potrebbe sembrare un problema, ma in realtà non lo è in quanto la mossa disponibile nella tabu search inserisce un 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 R. 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 R, è facile vedere come al diminuire di R diminuisce 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 tutte 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 ogni 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 il 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 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 la soddisfazione 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 consiste nel scegliere l'articolo minimo (risp. massimo) da aumentare e appartente al periodo per arrivare al minimo dichiarato dal bound
2. **Proporzionale:** consiste nel continuare a ciclare tutti gli articoli appartenti al periodo e aumentarli di una unità finchè 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 anticipare l'acquisto e non ordinare articoli in più.

Questi controlli 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 poichè alla fine l'algoritmo restituisce una soluzione ottima, che corrisponde alla miglior combinazione per ogni range di copertura di ogni articolo tra data d'ordine e fornitore, l'ottimo 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 sarebbe necessario 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 servizio e le sue funzionalità sono state sviluppate in un ordine preciso. Si è cercato di implementare il prodotto con un approccio incrementale, in modo tale da avere sempre un prodotto funzionante. In particolare, lo sviluppo è stato svolto nel seguente ordine:

* creazione della winform
* 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* veniva organizzata un 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
* 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 feature 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 *Algorithm-Strategy* e nella classe *Program* creare il corrispondente oggetto del nuovo algoritmo. Chiaramente, affinchè la classe sia istanziabile, è necessario implementare tutte le funzioni 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 e 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 portino alla creazione di un algoritmo che genera soluzioni migliori.
Grazie all'utilizzo del design pattern strategy è possibile estendere la classe astratta *GreedyAlgorithm*, quest'ultima 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 (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.
Per esempio se precedentemente la mossa veniva applicata a un solo codice articolo, si potrebbe pensare di applicarla a k codici articolo, con k deciso dal programmatore. Rimane comunque fondamentale mantenere la randomicità delle scelte.

Multi-threading
Questa estensione del progetto andrebbe a considerare la creazione di più 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 non viene effettuata alcuna ricombinazione, ma ogni soluzione viene 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 alle soddisfazione delle seguenti necessità e aspettative:

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

Di seguito vegono 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). Questi ultimi rendono più semplice la creazione di ambienti grafici per applicazioni ed elegante il risultato.

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. É 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

5.1 Verifica

5.1.1 Documentazione

5.1.2 Testing del modulo

5.2 Validazione

5.2.1 Documentazione

5.2.2 Codice

Capitolo 6

Conclusioni

6.1 Prodotto finale

6.2 Raggiungimento degli obiettivi

6.3 Conoscenze acquisite

6.4 Valutazione complessiva

Appendice A

Appendice A

Citazione

Autore della citazione

Glossario

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

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

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

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

UML in ingegneria del software *UML, Unified Modeling Language* (ing. linguaggio di modellazione unificato) è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. L'*UML* svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. 13

Bibliografia

Riferimenti bibliografici

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

Siti web consultati

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