POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Gestionale
Classe L8 – Ingegneria dell'Informazione

ANALISI E SIMULAZIONE PROGETTO BAG-SHARING

Relatore
Prof. Fulvio Corno

Candidato
Muraro Simone
259553

A.A. 2020/2021

Sommario

1. Proposta di progetto

1.1 Descrizione del problema proposto

Il Parlamento europeo e il Consiglio dell'Unione europea nel giugno 2019 hanno adottato la Direttiva sulla riduzione dell'incidenza di determinati prodotti di plastica sull'ambiente "Sup" (Single use plastics/plastica monouso) recepita dall'Italia con legge nazionale nell'aprile di quest'anno e in vigore da sabato 3 luglio 2021.
Il progetto bag sharing mira a risolvere il problema dei sacchetti di plastica monouso nella grande e media distribuzione, tramite dei sacchetti lavabili e riutilizzabili. Il programma, a partire dai dataset contenenti tutti i locali della grande e media distribuzione della regione Lombardia, crea un grafo e cerca il percorso migliore per il ritiro e la consegna dei sacchetti. Inoltre attraverso una simulazione si vuole stimare il consumo di sacchetti, il possibile riutilizzo di parte di questi e se il progetto sia economicamente conveniente.

1.2 Descrizione della rilevanza gestionale del problema

Il primo problema affrontato è di tipo logistico. Attraverso un algoritmo si vuole trovare la migliore ubicazione per i vari centri di ritiro e consegna dell'azienda bag-sharing; tramite la costruzione di grafi per ogni provincia e la ricerca del cammino migliore si riesce a stimare con precisione l'ubicazione dei centri provinciali. I centri provinciali coincideranno con la posizione del supermercato di partenza nella soluzione del cammino migliore. Trovati i centri provinciali si può creare un grafo con essi e cercare il cammino migliore per decidere dove situare l'azienda a livello regionale.
Il secondo problema è una simulazione del progetto bag sharing applicato alla realtà con l'obbiettivo finale di stimare delle variabili (costi/ricavi, fattibilità progetto, numero sacchetti totali) partendo da alcuni parametri (numero clienti giornalieri, media giorni ritorno al supermercato, costo sacchetti, costo lavaggio) alcuni settabili dall'utente, per simulare scenari diversi, altri fissi.

1.3 Descrizione dei data-set per la valutazione

Entrambi i dataset sono offerti dalla regione Lombardia tramite il portale OpenData nel quale sono raccolti tutti i dati inerenti a vari ambiti dell'organizzazione e gestione della regione.

Il primo dataset raccoglie tutte le Grandi strutture di vendita della regione Lombardia aggiornate all'11 Giugno 2021. Il dataset è stato creato per la prima volta a Maggio 2012 ed è scaricabile, sempre dal portale OpenData, la serie storica dei vari dataset aggiornati anno per anno ma per lo scopo del progetto è più utile il dataset più recente. Il dataset è composto da una sola tabella con 479 righe (ogni riga per una specifica struttura di vendita) e da 15 colonne che indicano vari attributi della specifica struttura di vendita.

Il secondo dataset raccoglie tutte le Medie strutture di vendita della regione Lombardia aggiornate all'11 Giugno 2021. Il dataset è composto da una sola tabella con 8.027 righe (una per ogni media struttura di vendita) e 14 colonne che specificano i vari attributi.

Per semplificare l'accesso dal programma ai due dataset è stato necessario riunirli in un unico database chiamato "strutture_vendita" contenente due tabelle distinte: "grandi_strutture" e "medie_strutture".

1.4 Descrizione preliminare degli algoritmi coinvolti

L'applicazione è realizzata attraverso il linguaggio java seguendo i pattern MVC (Model View Controller) e DAO (Data Access Object) distinguendo le parti di codice relative ad interfaccia, logica applicativa ed accesso al database, garantendo una migliore organizzazione e leggibilità del programma.
Gli algoritmi implementati sono di tipo ricorsivo e di simulazione.
L'algoritmo ricorsivo, a partire da un grafo contenente le varie strutture di vendita e le rispettive distanze, serve a calcolare il percorso migliore di consegna e ritiro tra le varie strutture.
L'algoritmo esplora i possibili cammini tra le varie strutture di vendita ed associa ad essi un costo dato dalla distanza percorsa per completare il percorso di tutte le strutture. Trovato il percorso migliore, il centro di raccolta/smistamento dei sacchetti coinciderà con la struttura di vendita di partenza del cammino.
Gli algoritmi riguardanti la simulazione sono implementati per verificare la fattibilità del progetto in termini di costi/ricavi e cercano di simulare al meglio la realtà tramite la possibilità di cambiare vari parametri della simulazione.

1.5 Descrizione preliminare delle funzionalità previste per l'applicazione software

L'applicazione presenta una sola interfaccia con quattro sezioni.
Nella prima sezione tramite una "choice box" (casella scelta multipla) si può selezionare il tipo di strutture di vendita da prendere in esame (medie o grandi).
La seconda sezione permette di scegliere le strutture di vendita di una determinata provincia o dell'intera regione (tramite una choice box) e tramite un bottone cercare la migliore ubicazione per un centro di smistamento/ritiro. Per arrivare a trovare la migliore posizione è necessario costruire un grafo che abbia come vertici le strutture di vendita e come peso degli archi la distanza tra le due strutture (calcolato come la distanza in linea retta tra due coordinate). Tramite un algoritmo ricorsivo si esplorano i vari cammini che toccano tutti i vertici del grafo e si sceglie quello di peso minore. La posizione del centro di smistamento/ritiro dei sacchetti lavabili coinciderà con la struttura di vendita di partenza nel cammino migliore. Il centro di smistamento regionale (selezionare nella choice box "intera regione") viene calcolato a partire da un grafo contenente i centri di smistamento di tutte le provincie (uno per provincia).
La terza sezione permette di settare dei parametri e tramite un bottone avviare la simulazione.
I parametri presi in considerazione sono: il costo dei sacchetti in tessuto, il costo di rivendita dei sacchetti, il costo dell'affitto giornaliero, il costo di lavaggio la durata della simulazione e il tempo di inter-arrivo dei nuovi clienti.
La simulazione si svolge prendendo in considerazione la giornata lavorativa di un negozio o supermercato (8h) e si calcolano il numero di nuovi sacchetti richiesti e il numero di quelli

riconsegnati da lavare. I costi relativi al ritiro e la consegna dei sacchetti sono calcolati seguendo il percorso migliore e moltiplicando la distanza percorsa per il costo della benzina al km. Il lavaggio tiene in considerazione dell'usura dei sacchetti (2 su 100 vanno buttati). Come output la simulazione restituisce i costi e i ricavi del progetto e il numero totale di sacchetti utilizzati. L'ultima sezione dell'applicazione è un'area di testo nella quale vengono visualizzati i risultati.

2. Descrizione dettagliata del problema affrontato

2.1 Logistica

La logistica è un settore chiave per molte aziende ed in continua evoluzione. A partire dagli anni 80 i computer hanno giocato un ruolo fondamentale nello sviluppo della logistica: all'inizio tramite semplici algoritmi poi attraverso software gestionali, come ERP. L'obbiettivo attuale è la logistica affiancata dalle tecnologie smart, dall'Internet of Things (IoT) e dall'intelligenza artificiale (AI).
Il problema logistico affrontato nell'applicazione è la scelta della posizione migliore per i centri di smistamento/ritiro dei sacchetti in modo da minimizzare la distanza da percorrere per la consegna o il ritiro. Minimizzando la distanza percorsa si riesce a risparmiare sotto molti aspetti fondamentali per un'azienda come il tempo di lavoro dell'operatore addetto alle consegne, l'usura del veicolo e i costi legati al carburante. Risparmiando 20 km al giorno si ottiene un risparmio di circa 500 euro in un anno per furgoncino impiegato. Per risolvere il problema è necessario costruire un grafo che contenga come vertici tutte le strutture di vendita e come peso degli archi la distanza tra le due strutture. La distanza viene calcolata a partire dalle coordinate geografiche delle due strutture di vendita tramite la formula di Heversine. L'algoritmo ricorsivo testa tutti i possibili percorsi e sceglie il percorso con peso minore. I percorsi per essere validi devono toccare tutti i vertici del grafo.

2.2 Simulazione

Per simulazione si intende un modello della realtà che consente di valutare e prevedere lo svolgersi dinamico di una serie di eventi susseguenti all'imposizione di certe condizioni da parte dell'utente. Le simulazioni sono uno strumento sperimentale molto potente e si avvalgono delle possibilità di calcolo offerte dall'informatica; la simulazione, infatti, non è altro che la trasposizione in termini logico-matematica-procedurali di un "modello concettuale" della realtà. Nell'ambito delle simulazioni, acquisisce notevole importanza la simulazione del funzionamento dei processi produttivi e logistici. Tali sistemi sono infatti caratterizzati da elevata complessità, numerose inter-relazioni tra i diversi processi che li attraversano, guasti dei segmenti, indisponibilità, stocasticità dei parametri del sistema. La simulazione, consentendo l'analisi della realtà ad un elevato livello di dettaglio e padroneggiando facilmente la complessità del sistema, fa si che alla fine sia possibile ottenere un gran numero di informazioni utili. Il prezzo da pagare per tale completezza è ovviamente il tempo; le operazioni di programmazione sono infatti assai lunghe, affinché si possano ottenere dei dati sufficientemente sensati e tali da dare la possibilità di ottenere un modello della realtà ad essa aderente.

L’applicazione, tramite la simulazione, mira a comprendere il rapporto costi/profitti tramite alcuni parametri settabili dall’utente, come il costo dei sacchetti, ed altri stocastici, come l’affluenza giornaliera o la percentuale di sacchetti riconsegnati.

3. Descrizione del data-set utilizzato per l’analisi

3.1 Strutture di vendita

Il database “strutture di vendita” riunisce i 2 dataset contenenti le grandi e medie strutture di vendita della lombardia.

3.1.1 Grandi strutture di vendita

Il dataset è composto da una sola tabella con 479 righe (ogni riga per una specifica struttura di vendita) e da 15 colonne:

-ID_PDV: Identificativo
-Provincia: Nome Provincia
-ISTAT: Codifica ISTAT del Comune
-COMUNE: Nome del Comune
-TITOLARE: Insegna del titolare della Partita Iva dell'esercizio commerciale
-CC: Centro Commerciale; possibili valori: SI' / NO
-INDIRIZZO: Indirizzo dell'esercizio commerciale
-INSEGNA
-Settoremerc.non alim. : Categorie per gli esercizi di tipo non alimentare;
Possibili valori: A=ABBIGLIAMENTO E SCARPE; B=ELECTRONICS AUDIO, VIDEO E TV; C=MOBILI E ACCESSORI CASE ARREDO; D=SPORT, TEMPO LIBERO, BRICO, LIBRI E CLASSIFICAZIONI RESIDUALI (COMPRENDENTE ANCHE AUTO);
-Sup.alim.: Superficie dedicata alla vendita di prodotti alimentari espressa in mq.
-Sup.non alim.: Superficie dedicata alla vendita di prodotti NON alimentari espressa in mq.
-Sup.totale: Superficie totale dedicata alla vendita
-geo_x: Coordinate espresse in Gradi rispetto al sistema di riferimento WGS_84
-geo_y: Coordinate espresse in Gradi rispetto al sistema di riferimento WGS_84
-location: Coordinate espresse in Gradi rispetto al sistema di riferimento WGS_84

3.1.2 Medie strutture di vendita

Il dataset è composto da una sola tabella con 8.027 righe (una per ogni media struttura di vendita) e 14 colonne:

-ID_PDV: Identificativo
-Provincia: Nome Provincia
-ISTAT: Codifica ISTAT del Comune
-COMUNE: Nome del Comune

-TITOLARE: Insegna del titolare della Partita Iva dell'esercizio commerciale
-CC: Centro Commerciale; possibili valori:SI' / NO
-INDIRIZZO: Indirizzo dell'esercizio commerciale
-INSEGNA
-Settoremerc.non alim. : Categorie per gli esercizi di tipo non alimentare;
Possibili valori: A=ABBIGLIAMENTO E SCARPE; B=ELECTRONICS AUDIO, VIDEO E TV; C=MOBILI E ACCESSORI CASE ARREDO; D=SPORT, TEMPO LIBERO, BRICO, LIBRI E CLASSIFICAZIONI RESIDUALI (COMPRENDENTE ANCHE AUTO);
-Sup.alim. : Superficie dedicata alla vendita di prodotti alimentari espressa in mq.
-Sup.non alim. : Superficie dedicata alla vendita di prodotti NON alimentari espressa in mq.
-Sup.totale: Superficie totale dedicata alla vendita
-geo_x: Coordinate espresse in Gradi rispetto al sistema di riferimento WGS_84
-geo_y: Coordinate espresse in Gradi rispetto al sistema di riferimento WGS_84
-location: Coordinate espresse in Gradi rispetto al sistema di riferimento WGS_84

4. Diagramma delle classi principali e descrizione degli algoritmi coinvolti

L'applicazione, sviluppata in linguaggio java, segue il pattern MVC (Model-View-Controller) ed il pattern DAO (Data-Access-Object) tramite i quali si tengono separati l'accesso ai dati del database, la parte algoritmica del programma e l'interfaccia utente.

4.1 I packages

I package sono lo strumento che meglio si presta a tenere separati vari ambiti dell'applicazione quindi sono presenti tre packages per separare il Model, il View e il Controller:

4.1.1 it.polito.tdp.BagSharing

Il package si occupa dell'interfaccia utente e della procedura di avvio dell'applicazione tramite tre classi:

`EntryPoint` necessario per l'avvio dell'applicazione stessa

`Main` gestisce avvio interfaccia nel quale verranno inseriti i dati necessari.

`FXMLController` Riceve dati in Input e collega l'interfaccia con il Model (logica dell'applicazione) ed è quindi la classe che interpreta i dati inseriti da utente e dopo un primo controllo permette lo svolgimento degli algoritmi fondamentali. La classe è destinata anche alla presentazione all'utente dei risultati degli algoritmi svolti.

4.1.2 it.polito.tdp.BagSharing.db

Tramite questo package si effettua la connessione e l'interrogazione del database tramite delle query SQL.
E' composto da due classi **DBConnect** e **BagSharingDAO**.

`DBConnect` Connessione con il database tramite il metodo di "connection pooling" che permette di risparmiare tempo di apertura e chiusura di una connessione.

`BagSharingDAO` Riceve parametri dal Model, spesso ricevuti a loro volta dall'FXMLController, e interroga il database tramite query SQL. Organizza i dati risultanti dalla ricerca in strutture dati opportune e le restituisce al Model.

4.1.3 it.polito.tdp.BagSharing.model

Il package contiene la logica applicativa quindi svolge la maggior parte delle operazione e dei calcoli.
È' composto da 4 classi: `Struttura`, `Simulator`, `Event`, `Model.`

`Struttura` Classe che identifica le strutture di vendita caratterizzate da 5 attributi e i metodi get() per poter accedere a tali attributi e il metodo toString() che restituisce una stringa contenente la descrizione della Struttura. Implementa inoltre i metodi HashCode() e Equals() per poter distinguere le varie strutture tra loro.

`Simulator` Classe destinata alle operazioni relative alla simulazione del progetto bagSharing.

`Event` Definisce la struttura, gli attributi e i diversi tipi di evento da utilizzare durante la simulazione.

`Model` Fulcro centrale dell'applicazione attraverso il quale tutti gli elementi vengono collegati. E' l'unica classe ad essere collegata sia all'interfaccia utente sia al database. All'interno del model sono definiti i metodi per creare un grafo, cercare una cammino di distanza minima, simulare il progetto ed altri metodi ausiliari.

4.2 Descrizione degli algoritmi utilizzati

L'applicazione si costituisce di quattro fasi principali durante le quali l'utente deve inserire degli input necessari al corretto funzionamento degli algoritmi.
La prima fase è l'avvio del programma, prima del quale l'utente deve aver caricato il dataset ed impostato i parametri di accesso ad HeidiSQL.
La seconda fase consiste nella creazione di un grafo e richiede la scelta delle dimensioni delle strutture di vendita e di un area geografica in cui si vuole effettuare la ricerca (Provincia o Intera regione).

La terza fase ricerca un percorso di lunghezza minima a partire dal grafo costruito precedentemente tramite un algoritmo ricorsivo. La ricerca del cammino può essere fatta a partire da una struttura, selezionata dall'utente, o provando tutte le strutture del grafo come partenza.
La quarta fase simula il progetto bag-sharing per tutte le strutture del grafo. L'utente può settare la durata della simulazione, il tempo di arrivo dei nuovi clienti e i costi relativi all'acquisto dei sacchetti dalla fabbrica, alla vendita dei sacchetti ai clienti, all'affitto dei sacchetti e al lavaggio. La simulazione restituisce i costi, i guadagni e il numero di nuovi sacchetti utilizzati.
Per la simulazione sono state fatte alcune semplificazioni che è bene menzionare:

- La giornata lavorativa di tutte le strutture di vendita inizia alle 8.00 e termina alle 20.00 nonostante ci siano strutture che seguono orari diversi

- Il costo del trasporto tiene conto del costo del carburante al Km e pertanto non tiene conto dell'usura del mezzo e dello stipendio dell'impiegato

- Il costo totale della simulazione non tiene conto degli stipendi degli impiegati, dell'acquisto dei veicoli per la consegna, dell'attrezzatura per il lavaggio (locale lavanderia e lavatrici)

- La distanza tra due strutture di vendita non tiene conto del percorso stradale ma è calcolata come distanza tra due punti sul globo terrestre

4.2.1 Fase1: Avvio Programma

Il pacchetto it.polito.tdp.BagSharing contiene le classi Main e EntryPoint, che si occupano di gestire l'avvio dell'applicazione e la classe FXMLController, la quale gestisce l'interazione con l'interfaccia utente.
L'interfaccia è realizzata tramite Scene Builder e si trova all'interno del pacchetto src/main/resources in formato fxml. La classe FXMLController contiene tutti i campi settabili dall'utente (selezione da un menù a tendina, input testuale), i metodi legati alla gestione delle operazioni effettuate (click di un bottone), gestisce tutti i possibili errori di input e restituisce il risultato del programma in output.
Durante la fase di avvio del programma viene creato un oggetto FXMLController e viene eseguito il metodo setModel() nel quale viene settato l'oggetto Model e si popolano le due combo box relative alla dimensione delle strutture e alle provincie da prendere in esame.
E' necessario aggiungere alla combo box delle provincie l'opzione "Intera regione" per permettere di calcolare il grafo dei centri provinciali della regione.

```
public void setModel(Model model) {
    this.model = model;

    this.boxDatabase.getItems().add("GRANDI STRUTTURE");
    this.boxDatabase.getItems().add("MEDIE STRUTTURE");
    this.boxProvincia.getItems().addAll(model.getAllProvincie());
    this.boxProvincia.getItems().add("INTERA REGIONE");
}
```

La classe FXMLController è composta da 3 metodi relativi alle tre azioni da svolgere: doGrafo per creare il grafo, doRicerca() per ricercare il cammino e doSimula() per simulare il progetto. I tre metodi si occupano di controllare i dati in input, richiamare l'algoritmo di risoluzione che si trova nel Model e infine presentare a video gli output della ricerca.
Un esempio è il metodo doSimulazione() nel quale si controllano i parametri inseriti dall'utente per la simulazione (costi,durata etc..), si controlla che sia stato creato il grafo su cui svolgere la simulazione e se tutto è settato correttamente si richiama l'algoritmo di risoluzione nel Model, altrimenti si mostrano dei messaggi di errore all'utente dove gli si indica una possibile risoluzione.

```
@FXML
void doSimulazione(ActionEvent event) {

    if(!this.txtCostoFabbrica.getText().isEmpty() && !this.txtCostoCliente.getText().isEmpty() &&  !this.txtCostoGG.getText().isEmpty() &&
            !this.txtCostoLavaggio.getText().isEmpty() && !this.txtDurata.getText().isEmpty() && !this.txtIntervallo.getText().isEmpty()) {

        if(model.getGrafo()!=null) {

            model.simula(this.txtCostoFabbrica.getText(), this.txtCostoCliente.getText(), this.txtCostoGG.getText(),
                         this.txtCostoLavaggio.getText(), this.txtDurata.getText(), this.txtIntervallo.getText());

            if(model.getCammino()!=null) {

                this.txtResult.setText("Costi "+model.getCostoProvincia()+"\nGuadagno: "+model.getGuadagnoProvincia()+
                        "\nNumero totale di sacchetti nuovi utilizzati"+model.getnBagTotaliProvincia()+"\ntenendo conto dei costi di trasporto");

            } else {

                this.txtResult.setText("Costi "+model.getCostoProvincia()+"\nGuadagno: "+model.getGuadagnoProvincia()
                    +"\nNumero totale di sacchetti nuovi utilizzati"+model.getnBagTotaliProvincia()+"\nSenza costi costi di trasporto");

                }
            } else {
                this.txtResult.setText("Scegliere l'area e creare il grafo prima di procedere con la simulazione");
            }
        } else {
            this.txtResult.setText("Inserire tutti i parametri della simulazione");
    }
}
```

4.2.2 Fase2: Creazione del grafo

L'utente sceglie dalla combo box boxDatabase se analizzare le grandi o medie strutture di vendita e sceglie l'area su cui si vuole creare il grafo tramite la combo box boxProvincia.
Alla pressione del bottone Grafo il programma esegue 5 metodi contenuti all'interno dell'FXMLController, del Model e di BagSharingDao:

- doGrafo(): E' contenuto nell'FXMLController ed è il primo metodo ad essere processato. Controlla che l'utente abbia settato le due combo box e in caso affermativo chiama il metodo creaGrafo del Model e restituisce in output il risultato, altrimenti restituisce all'utente un messaggio che indica di settare i campi mancanti.

```
@FXML
void doGrafo(ActionEvent event) {

    this.boxStruttura.getItems().clear();

    if(this.boxDatabase.getValue()!= null && this.boxProvincia.getValue()!= null) {

        model.creaGrafo(this.boxDatabase.getValue(),this.boxProvincia.getValue(),this.checkCC.isSelected(),this.checkAlim.isSelected());
        this.txtResult.setText("GRAFO CREATO\nVertici: " + model.numeroVertici() + "\nArchi: " + model.numeroArchi()+"\n");
        this.boxStruttura.getItems().addAll(model.getCentriProvinciali());

        } else if(this.boxDatabase.getValue()== null) {
            txtResult.setText("Selezionare dimensioni strutture \n");
        } else if(this.boxProvincia.getValue()== null) {
            txtResult.setText("Selezionare provincia in cui si vole effettuare la ricerca \n");
        }
}
```

- creaGrafo(): Il metodo è situato all'interno del Model ed il compito è quello di distinguere la procedura per la creazione del grafo per una specifica provincia oppure del grafo per l'intera regione.

```
public void creaGrafo(String database,String provincia, boolean CC,boolean supAlim) {

    if(provincia.equals("INTERA REGIONE")) {
        this.creaGrafoRegione(database, provincia, CC, supAlim);
        } else {
            this.creaGrafoProvincia(database, provincia, CC, supAlim);
    }

}
```

- creaGrafoProvincia(): Situato nel Model, è l'insieme di comandi necessari a costruire il grafo per una determinata provincia. La prima operazione da fare è distinguere tra le medie o grandi strutture e chiamare l'opportuno metodo della classe BagSharingDao per trovare le strutture da inserire come vertici del grafo.
  Il grafo costruito è un DefaultWeightedEdge ovvero un grafo non direzionato e pesato avente come vertici le strutture di vendita e come peso degli archi la distanza tra le strutture calcolata tramite il metodo Distance().

```
public void creaGrafoProvincia(String database,String provincia, boolean CC,boolean supAlim) {     //grafo per le provincie

    if(database.equals("GRANDI STRUTTURE"))                              //2 metodi diversi per interrogare il database poichè non corrispndevano alcunni parametri
        this.strutture= dao.getAllVerticiGrandi(provincia,CC,supAlim);

    if(database.equals("MEDIE STRUTTURE"))
        this.strutture= dao.getAllVerticiMedie(provincia,CC,supAlim);

    this.grafo = new SimpleWeightedGraph<>(DefaultWeightedEdge.class);

    Graphs.addAllVertices(grafo, strutture);                             //creazione grafo--> vertici presi dalla lista Strutture ottenuta interrogando il database


    for(Struttura s1: strutture) {                                       //creazione archi grafo
        for(Struttura s2: strutture) {
            if(!s1.equals(s2)) {                                         // peso dell'arco ottenuto tramite il metodo distance(lat1,lat2,lng1,lng2)
                Graphs.addEdge(grafo, s1, s2, distance(s1.getLat(),s2.getLat(),s1.getLng(),s2.getLng()));
            }
        }
    }

}
```

- creaGrafoRegione(): Il grafo della regione ha come vertici i centri provinciali uno per ogni provincia; Un centro provinciale coincide con la struttura di partenza nel cammino

migliore per quella provincia. Per costruire il grafo della regione è quindi necessario costruire un grafo per ogni provincia e trovare il cammino migliore. Salvata la città di partenza del cammino migliore in una lista chiamata centriProvinciali, si procede alla creazione del grafo della regione nello stesso modo di uno provinciale.

```
public void creaGrafoRegione(String database,String provincia, boolean CC,boolean supAlim) {    //grafo per la regione

    centriProvinciali= new ArrayList<Struttura>();         //lista contenente struttura di partenza nel cammino migliore

    for(String s: getAllProvincie()) {                     //Per ogni provincia

         creaGrafo(database,s,CC,supAlim);                 // creo il grafo
         List<Struttura> tmp=camminoMinimo(null);          // cerco cammino minimo

         if(tmp.size()>0)                                 //se trova un cammino minimo
             centriProvinciali.add(tmp.get(0));           //aggiungo la strttura di partenza del cammino minimo
    }

    this.grafo = new SimpleWeightedGraph<>(DefaultWeightedEdge.class);
    Graphs.addAllVertices(grafo, centriProvinciali);

        for(Struttura s1: centriProvinciali) {
            for(Struttura s2: centriProvinciali) {
                if(!s1.equals(s2)) {
                    Graphs.addEdge(grafo, s1, s2, distance(s1.getLat(),s2.getLat(),s1.getLng(),s2.getLng()));
                }
            }
        }
}
```

- getAllVerticiMedie() e getAllVericiGrandi(): I due metodi si trovano all'interno della classe BagSharingDao che è l'interfaccia con il database. Sono necessari due metodi distinti poiché i campi delle due tabelle differiscono sotto alcuni aspetti anche se il funzionamento dell'algoritmo è analogo. La query per interrogare il database cambia in base ai parametri settati dall'utente: la provincia è obbligatoria da selezionare, il campo CC seleziona le strutture situate nei Centri Commerciali e il campo SupAlim indica se sia presente un reparto alimentare. Effettuata la query si salvano le strutture all'interno di una lista utilizzata come valore di ritorno del metodo e successivamente passata al model grazie alla quale possiamo ottenere le strutture da inserire nel grafo come vertici.

```
public List<Struttura> getAllVerticiGrandi(String provincia,boolean CC, boolean supAlim){

    String sql = "SELECT DISTINCT ID_PDV,Provincia,Comune,Indirizzo,location FROM grandi_strutture WHERE Provincia=?";
    if(CC && supAlim) {
         sql = "SELECT DISTINCT ID_PDV,Provincia,Comune,Indirizzo,location FROM grandi_strutture WHERE Provincia=? AND CC=? AND SupAlim=? ";
    } else { if(CC) {
             sql = "SELECT DISTINCT ID_PDV,Provincia,Comune,Indirizzo,location FROM grandi_strutture WHERE Provincia=? AND CC=? ";
        } else { if(supAlim) {
                 sql = "SELECT DISTINCT ID_PDV,Provincia,Comune,Indirizzo,location FROM grandi_strutture WHERE Provincia=? AND SupAlim=?";
            }
        }
    }
    List<Struttura> result = new ArrayList<Struttura>();
    Connection conn = DBConnect.getConnection();
    try {
        PreparedStatement st = conn.prepareStatement(sql);
        st.setString(1, provincia);

            if(CC  && supAlim) {
                st.setString(2, "SI");
                st.setString(3, "0.00");
            } else if(CC)
                     st.setString(2, "SI");
                      else if(supAlim)
                         st.setString(2, "0.00");

        ResultSet res = st.executeQuery();
        while (res.next()) {
            result.add(new Struttura(res.getString("ID_PDV"),res.getString("Provincia"),res.getString("Comune"),res.getString("Indirizzo"),res.getString("location")));
        }
        res.close();
        st.close();
        conn.close();
        return result;
    } catch (SQLException e) {
        throw new RuntimeException("Database error", e) ;
    }
}
```

- distance(): il metodo è richiamato durante la procedura di creazione del grafo per settare il peso degli archi. Il metodo, situato nel Model, serve a calcolare la distanza tra due coordinate sul globo tramite la formula di Heversine e il risultato ottenuto indica la distanza “in linea d’area” espressa in Km.

```
public static double distance(double lat1,double lat2, double lon1, double lon2) {

    // The math module contains a function
    // named toRadians which converts from
    // degrees to radians.
    lon1 = Math.toRadians(lon1);
    lon2 = Math.toRadians(lon2);
    lat1 = Math.toRadians(lat1);
    lat2 = Math.toRadians(lat2);

    // Haversine formula
    double dlon = lon2 - lon1;
    double dlat = lat2 - lat1;
    double a = Math.pow(Math.sin(dlat / 2), 2)
    + Math.cos(lat1) * Math.cos(lat2)
    * Math.pow(Math.sin(dlon / 2),2);

    double c = 2 * Math.asin(Math.sqrt(a));

    // Radius of earth in kilometers. Use 3956
    // for miles
    double r = 6371;

    // calculate the result
    return(c * r);
}
```

4.2.3 Fase3: Ricerca cammino minimo

La ricerca del cammino minimo utilizza un algoritmo ricorsivo per trovare tutti i cammini possibili tra i vertici del grafo ed è quindi l’operazione più dispendiosa a livello computazionale. L’utente può selezionare una struttura dalla quale far partire la ricerca del cammino oppure può ricercare il percorso migliore provando tutte le strutture come partenza. Il primo caso si presta bene nel caso il Centro provinciale sia stato precedentemente collocato nella struttura di partenza ed è più rapido e meno dispendioso ma presenta il limite di non trovare il miglior percorso in assoluto ma bensì il miglior percorso a partire da una struttura selezionata. La ricerca a partire da tutti i centri risulta più dispendiosa ma è indispensabile per trovare il percorso migliore tra tutti i vertici del grafo, garantendo di collocare il centro provinciale nella migliore posizione. L’algoritmo ricorsivo si sviluppa attraverso due metodi del Model ed è composto da tre 4 fasi principali: l’aggiunta delle strutture, la chiamata ricorsiva, il caso intermedio e il caso terminale. Oltre ai 2 metodi del model dedicati all’algoritmo ricorsivo è presente un metodo dell’FXMLController per gestire l’input/output:

- doRicerca(): Viene chiamato alla pressione del bottone Ricerca Cammino e, trovandosi all’interno dell’FXMLController, si occupa dell’interfaccia con l’utente. Il metodo prima di iniziare la procedura di ricerca del cammino, contenuta nel model, deve assicurarsi che sia stato creato un grafo su cui svolgere la ricerca. Se il grafo è presente si può procedere

con l’algoritmo recursivo contenuto nel model, altrimenti viene mostrato un messaggio di avviso all’utente.

```
@FXML
void doRicerca(ActionEvent event) {


    if(model.getGrafo()!=null) {

        List <Struttura> res = model.camminoMinimo(this.boxStruttura.getValue());

        if (res.size()>0) {

        txtResult.appendText("Cammino trovato (peso = " + model.pesoCammino() + "):\n");

        for (Struttura s:res)
            txtResult.appendText(s + "\n");

        } else {
            txtResult.setText("Cammino non trovato.");
        }

    } else {
        txtResult.setText("Scegliere l'area e creare il grafo prima di clacolare il percorso minimo");
    }

}
```

- camminoMinimo(): All’inizio del metodo viene creata la lista adibita a contenere le Strutture nell’ordine in cui sono state attraversate nel caso del cammino migliore. Viene inoltre inizializzato il pesoMinimo ad un valore molto grande, la variabile deve tener traccia del peso del cammino migliore. La lista parziale serve a contenere le strutture attraversate fino a quel momento nella soluzione parziale.
  Nel caso l’utente non abbia selezionato nessuna Struttura di vendita dalla quale partire si provano tutte le strutture disponibili come partenza del cammino. La procedura consiste nell’aggiungere alla lista parziale una struttura, ricercare il percorso migliore a partire da quella struttura, ripulire la lista parziale ed aggiungere la struttura successiva fino a quando non sono state provate tutte le strutture come partenza.
  Se l’utente seleziona una Struttura da cui partire per la ricerca del cammino è necessario aggiungere alla lista parziale la struttura selezionata e richiamare la procedura ricorsiva che in questo caso ricerca il migliore cammino tra quelli che iniziano dalla struttura scelta.

```java
public List<Struttura> camminoMinimo(Struttura partenza){

    this.camminoMinimo = new ArrayList<Struttura>();
    this.pesoMinimo = 1000000.00;


    List<Struttura> parziale = new ArrayList<Struttura>();
                                //Inserimento Struttura
    if(partenza==(null)) {      //provo tutte le strutture come partenza
        for(Struttura s: grafo.vertexSet()) {
            parziale.add(s);
            recursive(parziale,0.00);

            parziale.clear();
        }
    } else {                    //seleziono struttura di partenza
            parziale.add(partenza);
            recursive(parziale,0);
    }


    return this.camminoMinimo;

}
```

- recursive(): Svolge gran parte dei calcoli dell'algoritmo ricorsivo e all'interno troviamo tre elementi fondamentali per permettere il corretto funzionamento. Il caso terminale è indispensabile per fermare la procedura ricorsiva e salvare in una variabile globale la soluzione parziale che altrimenti andrebbe persa. Un cammino parziale può entrare nel caso terminale solo se ha toccato tutti i vertici e il suo peso è minore del migliore cammino precedente, in caso contrario il cammino parziale entra nel caso intermedio all'interno del quale si effettuerà la chiamata ricorsiva. Nel caso intermedio si aggiunge alla lista parziale una tra le strutture collegate all'ultima struttura presente in lista e si calcola il costo del nuovo percorso, dopodiché si procede alla chiamata ricorsiva del metodo stesso recursive() passando come parametri il nuovo costo e la nuova lista parziale.

```java
private void recursive(List<Struttura> parziale, double pesoParziale) {

    if(parziale.size()==grafo.vertexSet().size()) {         //caso terminale
        if(pesoParziale < this.pesoMinimo) {

            this.pesoMinimo = pesoParziale;
            this.camminoMinimo = new ArrayList<Struttura>(parziale);

        }
    }

         //caso Intermedio
    for (Struttura s:Graphs.neighborListOf(grafo, parziale.get(parziale.size()-1))) {

        double pesoAggiuntivo = this.grafo.getEdgeWeight(this.grafo.getEdge(parziale.get(parziale.size()-1), s));

        if (!parziale.contains(s)) {

            pesoParziale += pesoAggiuntivo;
            parziale.add(s);
            recursive(parziale,pesoParziale);

            pesoParziale -= pesoAggiuntivo;
            parziale.remove(s);
        }
    }
```

4.2.4 Fase 4: Simulazione

La simulazione è l'obbiettivo principale dell'applicazione poiché compiere una tale ricerca senza l'ausilio della programmazione richiederebbe troppo tempo. La classe simulator è destinata alla logica della simulazione ed è articolata in 5 metodi principali oltre a diversi metodi get() utili al passaggio di varabili. La classe Event serve a definire gli oggetti che andranno a riempire la coda degli eventi della simulazione e in che ordine dovranno collocarsi. L'FXMLController mette a disposizione un metodo per far partire la simulazione:

- doSimulazione(): L'utente deve settare diversi parametri della simulazione: il costo dei sacchetti nuovi per l'azienda, il costo dei sacchetti nuovi ai clienti, il costo dell'affitto giornaliero di un sacchetto, il costo di lavaggio per un sacchetto, la durata della simulazione e la frequenza con cui arrivano nuovi clienti (espressa tramite un numero intero indicante i minuti tra un cliente nuovo e quello successivo).
  Controllati i dati in input dall'utente bisogna controllare che il grafo sia stato creato per poter simulare il progetto sulle strutture contenute in esso.
  Nel caso non si sia cercato precedentemente il percorso migliore per il grafo, la simulazione procede senza tenere conto dei costi legati al trasporto, se invece si è trovato il percorso migliore, la simulazione somma ai costi di gestione del progetto i costi di trasporto.
  Infine il metodo restituisce in output un messaggio di avviso se qualcosa non è andato a buon fine oppure i costi, i guadagni e il numero di nuovi sacchetti utilizzati durante la simulazione.

```
@FXML
void doSimulazione(ActionEvent event) {

    if(!this.txtCostoFabbrica.getText().isEmpty() && !this.txtCostoCliente.getText().isEmpty() &&  !this.txtCostoGG.getText().isEmpty() &&
            !this.txtCostoLavaggio.getText().isEmpty() && !this.txtDurata.getText().isEmpty() && !this.txtIntervallo.getText().isEmpty()) {

        if(model.getGrafo()!=null) {

            model.simula(this.txtCostoFabbrica.getText(), this.txtCostoCliente.getText(), this.txtCostoGG.getText(),
                         this.txtCostoLavaggio.getText(), this.txtDurata.getText(), this.txtIntervallo.getText());

            if(model.getCammino()!=null) {

                this.txtResult.setText("Costi "+model.getCostoProvincia()+"\nGuadagno: "+model.getGuadagnoProvincia()+
                        "\nNumero totale di sacchetti nuovi utilizzati"+model.getnBagTotaliProvincia()+"\ntenendo conto dei costi di trasporto");

            } else {

                this.txtResult.setText("Costi "+model.getCostoProvincia()+"\nGuadagno: "+model.getGuadagnoProvincia()
                    +"\nNumero totale di sacchetti nuovi utilizzati"+model.getnBagTotaliProvincia()+"\nSenza costi costi di trasporto");

                }
            } else {
                this.txtResult.setText("Scegliere l'area e creare il grafo prima di procedere con la simulazione");
            }
        } else {
            this.txtResult.setText("Inserire tutti i parametri della simulazione");
    }
}
```

- Simula(): Il metodo si trova nel Model e serve a controllare i dati immessi dall'utente e a far partire una simulazione per ogni centro, infine serve a prelevare i risultati della simulazione dalla classe Simulator e passarli all FXMLController per mostrarli a video all'utente.
  Per assicurarsi di non generare errori durante la conversione da Stringa, ricevuta tramite area di testo dall'utente, a numero decimale si ricorre all'utilizzo della struttura try{} catch() che permette di intercettare errori e gestirli (ad esempio mostrando un messaggio di avviso) evitando così il blocco del programma. Nel caso tutti i dati in input siano corretti, vengono inizializzati i parametri della simulazione tramite il metodo init() e viene fatta partire una simulazione tramite il metodo run() per ogni struttura presente nel grafo. I costi e i guadagni di ciascuna simulazione vengono sommati e successivamente si aggiungono i costi legati al traporto. Il trasporto si articola in due viaggi giornalieri, uno per il ritiro dei sacchetti da lavare e l'altro per la consegna dei sacchetti puliti e nuovi, i costi sono calcolati tenendo conto della distanza del percorso migliore moltiplicato per il costo della benzina al km.

```
public void simula(String costoSacchetti, String costoSacchettiCliente, String costoAffittoSacchetti,String costoLavaggio,String durata,String intervallo) {

    Simulator sim=new Simulator();
    this.costoProvincia=0.00;
    this.guadagnoProvincia=0.00;
    this.nBagTotProvincia=0;
    double costoTrasporto=0.00;
    costoTrasporto= (this.pesoCammino()*.10);

    try {
         double cS=Double.parseDouble(costoSacchetti);
         double cSC= Double.parseDouble(costoSacchettiCliente);
         double cAS= Double.parseDouble(costoAffittoSacchetti);
         double cL=Double.parseDouble(costoLavaggio);
         int duration=Integer.parseInt(durata);
         int interval= Integer.parseInt(intervallo);

         for(int i=0;i<=grafo.vertexSet().size();i++) {
                sim.init(cS,cSC,cAS,cL,duration, interval);
                sim.run();

                this.costoProvincia+=sim.getCostoStruttura();
                this.guadagnoProvincia+=sim.getGuadagnoStruttura();
                this.nBagTotProvincia+= sim.getnBagTotaliStruttura();
           }

                this.costoProvincia+=(costoTrasporto*duration*2);  // costo una tratta*giorniSimulazione*2 viaggi al giorno

    } catch(Exception e) {
        System.out.println("Exception: " + e);
    }

}
```

- Init(): metodo del Simulator che gestisce l'inizializzazione dei parametri della simulazione e l'inizializzazione delle variabili temporali necessarie per far partire e terminare la simulazione.

```
// Impostazione dei parametri iniziali

public void init(double costoSacchetti, double costoSacchettiCliente, double costoAffittoSacchetti,double costoLavaggio,int durata,int intervallo) {

    this.costoSacchetti=costoSacchetti;
    this.costoSacchettiCliente=costoSacchettiCliente;
    this.costoAffittoSacchetti=costoAffittoSacchetti;
    this.costoLavaggio=costoLavaggio;
    this.giornoInizio=LocalDate.now();
    this.giornoFine=giornoInizio.plus(durata, ChronoUnit.DAYS);
    this.inizio=LocalDateTime.of(giornoInizio, oraApertura);
    this.fine=LocalDateTime.of(giornoFine, oraChiusura);
    this.tIn=intervallo;
}
```

- Run(): Dopo aver settato le variabili che descrivono lo stato del sistema durante l'evoluzione della simulazione, si può iniziare a riempire la coda con gli eventi iniziali di tipo "Nuovo cliente" con cadenza scelta dall'utente e a fine giornata si inserisce un evento lavaggio per i sacchetti restituiti da lavare. Quando tutti gli eventi iniziali sono stati aggiunti alla coda si procede a processare gli eventi in ordine cronologico tramite il metodo processEvent().

```java
// Simulazione
public void run() {
    this.queue = new PriorityQueue<Event>() ;

    // Stato iniziale
    this.nBag = 0 ;
    this.nBagLavare=0;
    this.costo=0.00;
    this.guadagno=0.00;

    // Eventi iniziali
    LocalDateTime momento = this.inizio ;
    while(momento.isBefore(this.fine)) {

        if(momento.getHour()>=this.oraApertura.getHour() && momento.getHour()<this.oraChiusura.getHour()) {
            this.queue.add(new Event(momento, EventType.NUOVO_CLIENTE,this.getNCasualeBag(),0)) ;
            momento = momento.plus(this.tIn, ChronoUnit.MINUTES) ;
        }
         else {
            this.queue.add(new Event(momento.plus(5, ChronoUnit.MINUTES), EventType.LAVAGGIO,0,0)) ;
            momento= momento.plus(12, ChronoUnit.HOURS);
        }
    }

    // Ciclo di simulazione
    while(!this.queue.isEmpty()) {
        Event e = this.queue.poll();
        processEvent(e) ;
    }
}
```

- ProcesEvent(): il metodo svolge determinate azioni in base al tipo di evento preso in considerazione.
  L'evento "NUOVO_CLIENTE" prevede che meta dei nuovi clienti scelgano di non utilizzare il servizio bag-sharing, i motivi che portano a non utilizzare il servizio possono essere la preferenza di sacchetti monouso oppure l'impiego di sacchetti riutilizzabili propri. Il restante 50% accettano di entrare a far parte del progetto bag-sharing e possono acquistare o noleggiare un sacchetto tramite il metodo iscrittoBagSharing().
  I clienti abituali (evento "CLIENTI_ABITUALE"), a differenza dei nuovi clienti, aderiscono tutti al progetto bag-sharing e pertanto effettueranno o un acquisto o un noleggio sempre tramite il metodo iscrittoBagSharing().
  L'evento "RESTITUZIONE" modella il momento di riconsegna del sacchetto da parte dell'utente. La riconsegna prevede di restituire al cliente parte dei soldi pagati per l'acquisto momentaneo del sacchetto. L'importo da restituire viene calcolato sottraendo al costo del sacchetto pagato dal cliente il costo da pagare per l'affitto del sacchetto. Se il costo d'affitto risulta essere maggiore dell'acquisto per il cliente è chiaro che il cliente voglia tenersi la busta e quindi non ci sono costi di restituzione da sostenere. Effettuata la restituzione si deve aggiungere al numero di sacchetti da lavare, i sacchetti appena restituiti e si genera un nuovo evento "CLIENTE ABITUALE" poiché un cliente che si reca al supermercato per restituire un sacchetto sicuramente effettuerà una nuova spesa.
  Il "LAVAGGIO" viene posto a conclusione di ogni giornata della simulazione e prevede che vengano aggiunti i costi per il lavaggio dei sacchetti utilizzati. Prima del lavaggio si controlla lo stato attuale delle borse riconsegnate e si stima che il 2% di esse sia troppo logoro per poter essere lavato e rimesso in commercio, quindi viene scartato.

```
 // processEvent
private void processEvent(Event e) {
    switch(e.getType()) {

    case NUOVO_CLIENTE:
            double num = Math.random()*2 ;  // [0, 2)
            if (num<1.0) {                  //clienti che accettano di far parte del progetto bagSharing (50% dei nuovi clienti)
                this.iscrittoBagSharing(e);
                }                           // restanti 50% non accettano bagSharing
        break ;
    case CLIENTE_ABITUALE:
        this.iscrittoBagSharing(e);     //tutti i clienti abituali sono iscritti al progetto BagSharing
        break ;
    case RESTITUZIONE:
        double costoClienteAffitto=0.00;
        costoClienteAffitto=e.getnGG()*this.costoAffittoSacchetti;
        if(costoClienteAffitto< this.costoSacchettiCliente) {  //se il costo dell'affitto non supera il costo del sacchetto
            double costoDaSostenere=this.costoSacchettiCliente-costoClienteAffitto;
            costo+= costoDaSostenere*e.getnBagPreso();
            this.nBagLavare+= e.getnBagPreso();
            this.queue.add(new Event(e.getTime().plus(5,ChronoUnit.MINUTES), EventType.CLIENTE_ABITUALE, this.getNCasualeBag(), 0));
        }
        break ;
    case LAVAGGIO:
        this.nBagLavare-=this.nBagLavare*0.02;      //il 2% delle buste restituite sono logore e vanno buttate
        costo+=this.nBagLavare*this.costoLavaggio;
        this.nBag+=this.nBagLavare;                 //si aggiungono tutte le bag lavate alle bag disponibili
        this.nBagLavare=0;                          //alla fine del lavaggio non ci sono più bag da lavare
        break ;
    }
}
```

- iscrittoBagSharing(): Viene richiamato dal procesEvent nel caso un cliente (sia nuovo che abituale) decida di far parte del progetto ed acquistare o noleggiare un sacchetto. Il 50% dei clienti che decide di far parte del progetto bagSharing acquista uno o più sacchetti la prima volta, tramite il metodo vendiSaccchetto, e successivamente non avrà più bisogno del progetto bagSharing. La restante metà noleggia i sacchetti ogni volta che si reca a fare la spesa e anche in questo caso il cliente paga il sacchetto a prezzo pieno, tramite il metodo vendiSacchetto, ma viene aggiunto un nuovo evento Restituzione che gli permetterà di avere indietro parte dei soldi del sacchetto. La restituzione può avvenire ad intervalli diversi per ogni cliente ed è suddivisa nel seguente modo: il 10% dopo 2 giorni, il 15% entro tre giorni, 20% dopo 4 giorni e un altro 20% dopo 5, 15% restituisce i sacchetti dopo 6 giorni, 10% dopo 7, 5% dopo 8 e l'ultimo 5% al nono giorno.

```
private void iscrittoBagSharing(Event e) {

    double numSi = Math.random()*2 ;

    if(numSi<1.0) {                                 //50% dei clienti compra la busta e non la vuole affittare
        this.vendiSacchetto(e.getnBagPreso());
    } else {                                        //50% dei clienti affitta bag
        double numGG = Math.random()*100 ;

        if(numGG<10) {                              //10% la restituisce entro 2 giorni

            this.vendiSacchetto(e.getnBagPreso());
            if(e.getTime().plus(2, ChronoUnit.DAYS).isBefore(fine))
            this.queue.add(new Event(e.getTime().plus(2, ChronoUnit.DAYS),EventType.RESTITUZIONE,e.getnBagPreso(),2));

        } else {
            if(numGG<25) {                          //15% la restituisce entro 3 giorni
                this.vendiSacchetto(e.getnBagPreso());
                if(e.getTime().plus(3, ChronoUnit.DAYS).isBefore(fine))
                this.queue.add(new Event(e.getTime().plus(3, ChronoUnit.DAYS),EventType.RESTITUZIONE,e.getnBagPreso(),3));
            } else {
                if(numGG<45) {                      //20% la restituisce dopo 4 giorni
                    this.vendiSacchetto(e.getnBagPreso());
                    if(e.getTime().plus(4, ChronoUnit.DAYS).isBefore(fine))
                    this.queue.add(new Event(e.getTime().plus(4, ChronoUnit.DAYS),EventType.RESTITUZIONE,e.getnBagPreso(),4));
                } else {
                    if(numGG<65) {
```

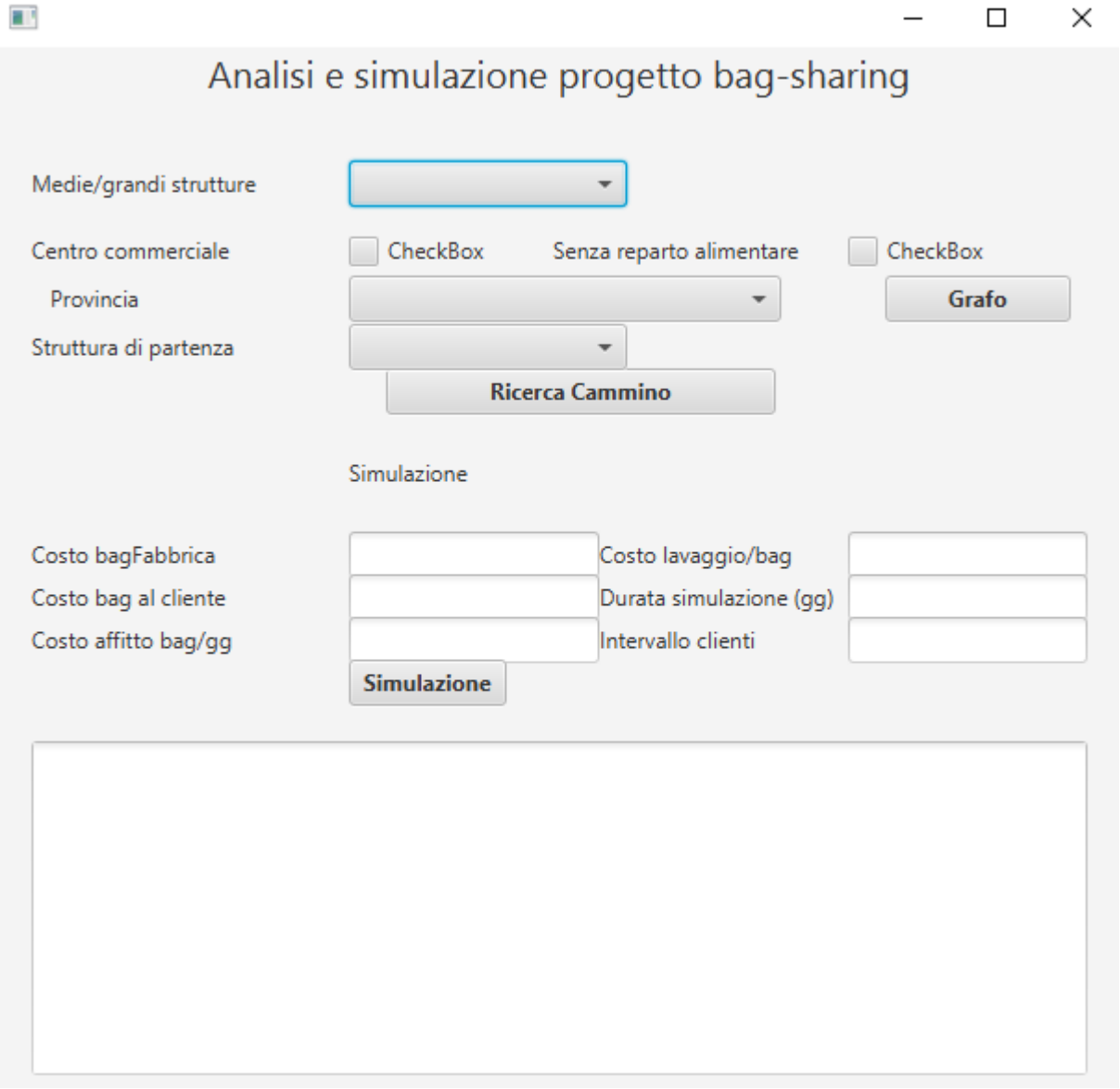
- vendiSacchetto(): Grazie a questo metodo si vendono i sacchetti ai clienti e vengono aggiornate le variabili della simulazione che saranno restituite come risultato. Durante la vendita possono presentarsi situazioni diverse in base alla disponibilità di sacchetti lavati o novi e in base al numero di sacchetti presi dal cliente (calcolati casualmente tramite il metodo getNumeroSacchetti() che restituisce un numero intero tra 1 e 4). Se le borse già lavate coprono il fabbisogno del cliente, l'azienda guadagna il nuovo costo per comprare le borse senza dover spendere per comprarne altre dalla fabbrica, al contrario se non si dispone di borse lavate l'azienda deve sostenere il costo di acquisto delle nuove borse e rivenderle al cliente da cui incassa il prezzo di rivendita. Se le borse lavate non coprono l'intero fabbisogno ma solo una parte della richiesta del cliente, si possono affittare tutte le borse disponibili già lavate e coprire la restante parte con borse nuove.

```
private void vendiSacchetto(int numeroSacchettiPreso) {

    if(this.nBag>=numeroSacchettiPreso) {                    // se disponibilità bag sufficiente non devo comprare una nuova bag
         guadagno=guadagno+(costoSacchettiCliente*numeroSacchettiPreso);   //ho solo guadagno e tolgo una bag da quelle disponibili
         nBag--;

     } else {
         if(this.nBag==0) {   // controllo disponibilità bag, se =0, utilizzo una busta nuova
         costo=costo+(costoSacchetti*numeroSacchettiPreso);
         guadagno=guadagno+(costoSacchettiCliente*numeroSacchettiPreso);
         nBagTot+= numeroSacchettiPreso;      //aumento il numero di nuove borse utilizzate
            } else {                                          //alcune bag disponibili/alcune bag da comprare
                int numeroSacchettiNonDisponibile=numeroSacchettiPreso-nBag;
                costo=costo+(costoSacchetti*numeroSacchettiNonDisponibile);
                 guadagno=guadagno+(costoSacchettiCliente*numeroSacchettiPreso);
                 nBagTot+= numeroSacchettiNonDisponibile;
                 nBag=0;
            }
         }

}
```

5 Interfaccia applicazione

L'interfaccia dell'applicazione guida l'utente nei vari passaggi tramite i quali si vuole arrivare allo sviluppo di unna simulazione.
L'utente come prima cosa deve scegliere se prendere in considerazione le grandi strutture di vendita oppure le medie strutture di vendita, questa scelta influenza il carico computazionale del programma poiché nel primo caso si dispone di un dataset relativamente contenuto (solo 476 strutture) mentre nel secondo caso le strutture presenti nel dataset sono poco piu di 8000.
La seconda scelta necessaria prima di premere il tasto crea Grafo e procedere con la creazione di esso, è la selezione di un'area geografica da cui prendere le strutture per il grafo e questa può coincidere con una provincia o con l'intera regione. Se il grafo risulta essere troppo esteso si rischia che l'algoritmo per la ricerca del cammino minimo non giunga a termine entro un tempo utile, quindi si possono filtrare le strutture da inserire tramite due checkBox che indicano se la struttura sia situata in un centro commerciale e se la struttura sia dotata di reparto alimentare.
Settate le prime due combo box e facoltativamente le due chack box si può procedere alla creazione del grafo e al riempimento della terza combo box con le strutture presenti come vertici del grafo appena creato. L'utente a questo punto può già far partire la ricerca del cammino minimo premendo il bottone Cerca cammino oppure può settare, tramite la terza combo box, la

struttura da cui far partire la ricerca del cammino minimo. Anche in questo caso la scelta condiziona il carico computazionale che nel primo caso risulta essere molto più elevato visto che si provano molti più percorsi possibili. Dopo aver costruito il grafo e calcolato il percorso migliore, l'utente può effettuare la simulazione. Per il corretto funzionamento dell'algoritmo ricorsivo, l'utente deve settare 6 variabili: 4 riguardanti i costi del progetto (sacchhetti nuovi, sacchetti al cliente, affitto e lavaggio) e 2 riguardanti variabili temporali come la durata della simulazione e il tempo di inter-arrivo tra un cliente e quello successivo. Settati tutti i parametri è possibile premere il bottone Simula e far partire la simulazione. I risultati delle operazioni compiute dall'utente (la pressione dei bottoni) e quindi degli algoritmi svolti dal programma vengono mostrati nell'area di testo a fondo schermata, nella quale vengono anche mostrati eventuali messaggi di avviso o di errore.
Link al video dimosrativo:



6. Risultati sperimentali ed esempi

6.1 Esempio 1

Dimensione delle strutture di vendita: *grandi strutture di vendita*
Provincia: *Sondrio*
Centro commerciale: *non selezionato* (strutture situate in centro commerciale e all'esterno)
Senza reparto alimentare: *non settato* (strutture con e senza reparto alimentare)

Grafo

Il grafo creato ha 9 vertici corrispondenti alle grandi strutture di vendita situate nella provincia di Sondrio ed ha 36 archi. Il grafo costruito è quindi un grafo connesso e completo pertanto il numero di archi del grafo è facilmente ottenibile a partire dal numero di vertici tramite la formula $N(N-1)/2$ dove N è il numero di vertici.
Nell'esempio N=9 ed il numero di arci corrisponde a (9*8)/2=36

Ricerca del cammino

In questo esempio effettuiamo una ricerca del percorso migliore provando tutte le strutture come partenza del cammino. L'operazione è molto dispendiosa a livello computazionale e giunge a termine in un tempo ragionevole per grafi con dimensioni inferiori agli 11 vertici.
Il cammino trovato, come previsto, attraversa tutte le strutture di vendita del grafo ed è quello di peso minore.

```
GRAFO CREATO
Vertici: 9
Archi: 36
Cammino trovato (peso = 73.98764736364443):
Struttura [id=9879, provincia=SO, comune=Prata Camportaccio, indirizzo=VIA GIULIO CHIARELLI 4-6]
Struttura [id=2654, provincia=SO, comune=Prata Camportaccio, indirizzo=VIA NAZIONALE, 36]
Struttura [id=3358, provincia=SO, comune=Piantedo, indirizzo=VIA LA ROSA 354]
Struttura [id=3690, provincia=SO, comune=Rogolo, indirizzo=VIA ANDREA DORIA N. 2]
Struttura [id=200411, provincia=SO, comune=Morbegno, indirizzo=VIALE STELVIO 300]
Struttura [id=8836, provincia=SO, comune=Talamona, indirizzo=VIA STELVIO, 2/B]
Struttura [id=3692, provincia=SO, comune=Castione Andevenno, indirizzo=VIALE BRUNO TIRELLI 2]
Struttura [id=14719, provincia=SO, comune=Castione Andevenno, indirizzo=VIA DEL PIANO]
Struttura [id=13136, provincia=SO, comune=Bianzone, indirizzo=VIA COLOMBINI 25]
```

Simulazione

I parametri settati cercano di porre la simulazione in un contesto più realistico possibile. I sacchetti in tessuto sono venduti mediamente tra 0.50€ e 5€ al dettaglio ma trattandosi di un ingente quantitativo si può calcolare un prezzo stock di 0.40€ a sacchetto. Il costo al cliente non può essere eccessivamente alto poiché si scoraggerebbe l'utilizzo del nuovo servizio ed inoltre l'obbiettivo principale del progetto è guadagnare tramite l'affitto e non tramite la vendita dei sacchetti. Il costo dell'affitto dei sacchetti è uno dei parametri più influenti sulla simulazione ed è il più difficile da settare correttamente, poiché un valore troppo basso farebbe guadagnare troppo

poco mentre un valore troppo alto costringerebbe la maggior parte dei clienti ad acquistare i sacchetti al posto che affittarli.
Il costo del lavaggio per sacchetto viene calcolato tenendo conto che un ciclo di una lavatrice da 10kg costa circa 1 € e in un singolo ciclo vengono lavate 100 borse con un peso di 100g l'una.
La durata può essere scelta a piacere tenendo sempre presente che una durata della simulazione troppo breve non permette la creazione di tutti gli eventi rendendo la simulazione poco significativa.
L'utente può settare liberamente il tempo di inter-arrivo dei clienti ma in corrispondenza di un tempo elevato, la simulazione impiegherà più tempo per dare risultati soddisfacenti.

Costo dei sacchetti nuovi all'azienda:	0.40€
Costo dei sacchetti nuovi al cliente:	0.50€
Costo dell'affitto dei sacchetti al giorno:	0.10€/ al giorno per ogni sacchetto
Costo lavaggio sacchetti:	0.01€/saccchetto
Durata simulazione:	30 giorni

```
Costi 9599.825884181857
Guadagno: 17313.0
Numero totale di sacchetti nuovi utilizzati: 106772
tenendo conto dei costi di trasporto
```

Costo dei sacchetti nuovi all'azienda:	0.40€
Costo dei sacchetti nuovi al cliente:	0.50€
Costo dell'affitto dei sacchetti al giorno:	0.05€/ al giorno per ogni sacchetto
Costo lavaggio sacchetti:	0.01€/saccchetto
Durata simulazione:	30 giorni

```
Costi 8674.01588418187
Guadagno: 23576.0
Numero totale di sacchetti nuovi utilizzati: 41328
Tenendo conto dei costi di trasporto
```

Valutazione:
La creazione del grafo avviene rapidamente e anche il percorso migliore viene trovato in meno di un secondo, sia nel caso si provino tutte le strutture come partenza sia che se ne scelga una in particolare.
Le due simulazioni restituiscono risultati molto diversi nonostante sia stato cambiato solamente il costo di affitto dei sacchetti. Il prezzo di affitto nella seconda simulazione viene ridotto, quindi i il servizio diventa più conveniente per i clienti, ma la diminuzione causa inaspettatamente un aumento dei guadagni per l'azienda ed a una diminuzione dei costi. Questo comportamento è spiegabile guardando i numeri delle nuove borse utilizzate: nel primo caso sono circa il doppio del secondo e questo vuol dire che molti clienti, siccome pagherebbero di più di affitto nel

momento della restituzione rispetto all'acquisto del sacchetto, preferiscono la seconda opzione, riducendo così la principale fonte di guadagno del progetto che si basa sull'affitto.

6.2 Esempio 2

Dimensione delle strutture di vendita:	*grandi strutture di vendita*
Provincia:	*Intera regione*
Centro commerciale:	*selezionato* (solo strutture situate in centro commerciale)
Senza reparto alimentare:	*selezionato* (strutture senza reparto alimentare)

Grafo

Il grafo viene creato dopo aver cercato le strutture di partenza del cammino migliore per ogni provincia ed ha 8 vertici e 28 archi. Le provincie della Lombardia sono 12 e pertanto dovrebbero esserci altrettanti vertici nel grafo, uno per ogni Centro Provinciale, ma non tutte le provincie presentano strutture che rispettino le condizioni impostate (centro commerciale e superficie alimentare). L'algoritmo è molto dispendioso a livello computazionale poiché ricerca il percorso migliore per ogni provincia e l'algoritmo ricorsivo alla base della ricerca è molto complesso e lungo. La ricerca dei centri provinciali per l'intera regione può essere perciò applicata solo in caso di un numero ristretto di strutture per ogni provincia.

Ricerca del cammino

La ricerca del cammino minimo viene svolta a partire dai centri migliori di ogni provincia e l'algoritmo non subisce cambiamenti rispetto al primo esempio. L'unico differenza riscontrabile è la distanza del percorso che, tenendo conto di tutta la regione, è più lunga rispetto alla distanza per una provincia.

```
GRAFO CREATO
Vertici: 8
Archi: 28
Cammino trovato (peso = 295.2839182281687):
Struttura [id=5793, provincia=MN, comune=Bagnolo San Vito, indirizzo=LOCALITA BASSE DI MEZZO]
Struttura [id=4795, provincia=BS, comune=Lumezzane, indirizzo=VIA S. NICOLA DA TOLENTINO, 25]
Struttura [id=14719, provincia=SO, comune=Castione Andevenno, indirizzo=VIA DEL PIANO]
Struttura [id=14807, provincia=BG, comune=Orio al Serio, indirizzo=VIA PORTICO, 12]
Struttura [id=201152, provincia=LC, comune=Osnago, indirizzo=VIA DELLE MARASCHE]
Struttura [id=1337, provincia=CO, comune=Cermenate, indirizzo=VIA EUROPA UNITA]
Struttura [id=2415, provincia=MI, comune=Rho, indirizzo=VIA CAPUANA ANG CORSO EUROPA]
Struttura [id=200781, provincia=VA, comune=Vedano Olona, indirizzo=LOCALITA' FONTANELLE]
```

Simulazione

La simulazione per l'intera regione esegue lo stesso algoritmo che si utilizza per la singola provincia, pertanto considera i Centri Provicniali come semplici strutture, non come centri di raccolta. I risultati della simulazione per l'intera regione sono quindi poco significativi per capire l'andamento del progetto a livello regionale ma se si volesse arrivare ad un grado di precisione maggiore, la simulazione diventerebbe insostenibile a livello computazionale.

Valutazione

La costruzione del grafo per l'intera regione è un processo notevolmente più complesso della costruzione di un grafo per la provincia ma è fondamentale per comprendere dove situare i Centri provinciali e calcolare la distanza tra essi. L'algoritmo richiama per ogni provincia la costruzione del grafo e il calcolo del cammino minimo quindi si avrà una soluzione in tempo ragionevole solo se ognuno dei grafi provinciali contiene un numero esiguo di strutture (non più di 10) quindi è necessario filtrare i vertici del grafo.

6.3 Esempio 3

Dimensione delle strutture di vendita:	*grandi strutture di vendita*
Provincia:	*Lecco* (LC)
Centro commerciale:	*selezionato* (solo strutture situate in centro commerciale)
Senza reparto alimentare:	*non settato* (strutture con o senza reparto alimentare)

Grafo

Il grafo viene creato senza problemi con 10 vertici corrispondenti alle grandi strutture di vendita della provincia di Lecco che si trovano all'interno di un centro commerciale e seguendo la regola dei grafi completi ha 45 archi.

Cammino minimo

Il cammino viene fatto partire da una struttura scelta dall'utente, per l'esempio si parte dalla struttura con id=13315 (provincia=LC, comune=Costa Masnaga, indirizzo=VIA PARADISO) e si vuole confrontare il peso del percorso migliore ottenuto con il peso del miglior cammino in assoluto per la provincia.

Cammino migliore a partire dalla struttura 13315

```
Cammino trovato (peso = 42.38757364872491):
Struttura [id=13315, provincia=LC, comune=Costa Masnaga, indirizzo=VIA PARADISO]
Struttura [id=5291, provincia=LC, comune=Casatenovo, indirizzo=V. CASATI, 28]
Struttura [id=201152, provincia=LC, comune=Osnago, indirizzo=VIA DELLE MARASCHE]
Struttura [id=464, provincia=LC, comune=Cernusco Lombardone, indirizzo=VIA SPLUGA 115]
Struttura [id=14412, provincia=LC, comune=Merate, indirizzo=VIA BERGAMO 19]
Struttura [id=10094, provincia=LC, comune=Barzago, indirizzo=VIA XXV APRILE 1]
Struttura [id=10890, provincia=LC, comune=Civate, indirizzo=VIA PAPA GIOVANNI XXIII VIA ALLA SANTA]
Struttura [id=7207, provincia=LC, comune=Pescate, indirizzo=VIA ROMA 7]
Struttura [id=1612, provincia=LC, comune=Lecco, indirizzo=CORSO CARLO ALBERTO, 120]
Struttura [id=1611, provincia=LC, comune=Lecco, indirizzo=VIA AMENDOLA, 119]
```

Cammino migliore in assoluto per la provincia

```
GRAFO CREATO
Vertici: 10
Archi: 45
Cammino trovato (peso = 35.665975936934494):
Struttura [id=14412, provincia=LC, comune=Merate, indirizzo=VIA BERGAMO 19]
Struttura [id=464, provincia=LC, comune=Cernusco Lombardone, indirizzo=VIA SPLUGA 115]
Struttura [id=201152, provincia=LC, comune=Osnago, indirizzo=VIA DELLE MARASCHE]
Struttura [id=5291, provincia=LC, comune=Casatenovo, indirizzo=V. CASATI, 28]
Struttura [id=10094, provincia=LC, comune=Barzago, indirizzo=VIA XXV APRILE 1]
Struttura [id=13315, provincia=LC, comune=Costa Masnaga, indirizzo=VIA PARADISO]
Struttura [id=10890, provincia=LC, comune=Civate, indirizzo=VIA PAPA GIOVANNI XXIII VIA ALLA SANTA]
Struttura [id=7207, provincia=LC, comune=Pescate, indirizzo=VIA ROMA 7]
Struttura [id=1612, provincia=LC, comune=Lecco, indirizzo=CORSO CARLO ALBERTO, 120]
Struttura [id=1611, provincia=LC, comune=Lecco, indirizzo=VIA AMENDOLA, 119]
```

Valutazione

Nell'esempio si vuole mettere in risalto la differenza tra la ricerca del cammino migliore in assoluto rispetto al cammino migliore data una struttura di partenza. Si può notare che nel caso si scelga una struttura di partenza, il cammino migliore trovato differisce dal cammino migliore in assoluto e il peso risultante sarà sempre maggiore o uguale al peso del cammino migliore in assoluto. La differenza di distanza da percorrere tra i due casi dell'esempio è di circa 10 km che potrebbero sembrare ininfluenti ma la spesa mensile aggiuntiva si attesterebbe intorno ai 50€, senza contare il tempo risparmiato dall'operatore. Il prezzo da pagare per compiere una ricerca che trovi il migliore cammino assoluto è una maggiore difficoltà computazionale e quindi un maggiore dispendio di tempo per portare a termine la richiesta.

6. Valutazioni finali e conclusioni

L'applicazione sviluppata riesce nell'intento di costituire uno strumento utile all'analisi del progetto bag sharing. Gli algoritmi sviluppati sono in grado di compiere ricerche che sarebbero troppo lunghe per un calcolo manuale.
Analizzando gli obiettivi posti in fase di progetto ed elencati nel Capitolo 1 si nota come essi siano stati soddisfatti:

Ricerca delle strutture dell'area: la realizzazione del grafo permette di rendersi conto del numero delle strutture presenti e quindi proporzionare le risorse per l'area selezionata

Efficienza nella consegna: attraverso la ricerca del cammino minimo si riesce ad ottimizzare il processo di consegna e ritiro dei sacchetti ed a stimare correttamente la posizione del centro Provinciale.

Simulazione: l'utente può verificare la convenienza del progetto per diverse configurazioni dei parametri e può stimare la combinazione migliore di parametri per far funzionare il progetto

Tuttavia i limiti dell'applicazione sono tanti e come già elencati nel Capitolo 3 sono numerosi i miglioramenti che possono essere applicati.

Non tenendo conto del magazzino, delle risorse umane e di alcuni costi, l'applicazione si concentra solo nel creare simulazioni adeguate alle caratteristiche descritte, lasciando all'utente la facoltà di aggiungere queste variabili in seguito.
Un'altra criticità è legata al tempo impiegato dall'applicazione per svolgere le operazioni.
In particolar modo il processo di ottimizzazione logistica che, sfruttando la ricorsione per la ricerca del cammino minimo, richiede una complessità computazionale non indifferente e perciò richiede l'utilizzo di variabili contenute e di tempo per portare a termine tutto il processo.
Per un miglioramento futuro dell'applicazione perciò è consigliato partire dalla velocità di esecuzione dell'algoritmo ricorsivo, magari imponendo più vincoli intermedi nella ricerca del risultato migliore. La simulazione potrebbe essere implementata attraverso l'introduzione di nuovi costi più dettagliati e di nuove opzioni come la realizzazione di abbonamento.