

POLITECNICO DI TORINO

Dipartimento di Ingegneria Gestionale e della
Produzione

Corso di Laurea in Ingegneria Gestionale
Classe L-8 Ingegneria dell'Informazione



Prova Finale EV Manager Tool per la Gestione della Mobilità Elettrica

Relatore
Prof. Fulvio Corno

Candidato
Baldo Alessandro
(s236651)

Anno Accademico 2018/2019

Indice

Proposta di progetto	3
Descrizione dettagliata del problema	5
Descrizione del data-set utilizzato per l'analisi.....	7
Descrizione ad alto livello delle strutture dati e degli algoritmi utilizzati.....	9
Diagramma delle classi delle parti principali dell'applicazione.....	12
Alcune videate dell'applicazione	13
Tabelle con risultati sperimentali ottenuti	15
Valutazione risultati ottenuti e conclusioni	17

Proposta di progetto

Studente proponente

Baldo Alessandro (s236651)

Titolo della proposta

EV Manager – Tool per la Gestione della Mobilità Elettrica

Descrizione del problema proposto

L'applicazione si propone di sensibilizzare l'utenza automotive, simulando, in prima istanza, una semplice analisi di mercato tra gli autoveicoli elettrici presenti tuttora nel mercato e debuttanti entro i prossimi 24 mesi. La seconda parte del programma permette di pianificare una rotta all'interno dello stato della California (ottimo compromesso per la sua morfologia per testare le prestazioni in termini di autonomia della batteria), tenendo conto delle soste obbligatorie presso le stazioni di ricarica elettrica presenti sul territorio.

Descrizione della rilevanza gestionale del problema

La mobilità a impatto zero rappresenta un tema che, negli anni a venire, diventerà centrale nel settore dell'autoveicolo. Il tool sfrutterà semplici principi di Ricerca Operativa (teoria dei grafi, cammino minimo, albero ricoprente) applicati al mondo della programmazione per offrire, in un contesto semplificato ma intuitivo, un primo approccio a un settore, sicuramente in crescita, ma ancora acerbo (sia lato software, sia lato trasporti).

A discrezione dell'utente sarà, ove possibile, effettuare il calcolo di un percorso ottimizzato sulla distanza (e sul numero di soste obbligatorie) tra i punti di partenza e destinazione scelti, basati sul modello di autoveicolo, che più ha soddisfatto le proprie preferenze.

Descrizione dei data-set

L'applicazione si appoggia su tre tipologie di data set:

- Un database contenente le informazioni su tutte le stazioni di ricarica elettriche, aggiornato ad aprile 2019, nello stato della California (fonte: https://afdc.energy.gov/fuels/electricity_locations.html#/analyze?region=US-CA&fuel=ELEC&ev_levels=all&show_map=true)
- Un database contenente l'elenco di tutte le città nello stato della California con informazioni sulle coordinate geografiche (fonte: https://www.kaggle.com/camnugent/california-housing-feature-engineering#cal_cities_lat_long.csv)
- Un database contenente l'elenco di tutti gli autoveicoli elettrici presenti sul mercato e la cui uscita è programmata entro il 2021, creato personalmente a partire dalla fonte: ev-database.org/ev-database.uk

Descrizione preliminare degli algoritmi coinvolti

Il programma sfrutterà concetti di Teoria sui Grafi, Cammino Minimo, Albero Ricoprente, Ricorsione e Trigonometria nella costruzione del grafo e del relativo calcolo del percorso ottimale. A supporto dei suddetti algoritmi, si ravvisa l'uso di pattern come l'MVC, DAO e ORM. Le librerie impiegate sono la JGraphT, la LatLng, la Hikari, MySQLConnector.

Descrizione preliminare delle funzionalità previste per l'applicazione software

Il software si articola in due parti. In prima istanza l'utente, attraverso opportuni indicatori, sceglie il modello di autoveicolo che più soddisfa le proprie preferenze, avendo indicazioni anche in termini di prezzo di vendita e/o prezzo di noleggio.

In secondo luogo, scelto il migliore match tra i risultati ottenuti al punto precedente, sarà data la possibilità di simulare un percorso all'interno dello stato della California. In tale simulazione verrà tenuto conto delle possibili soste obbligatorie nel tragitto, basate sull'autonomia nominale del mezzo e in aggiunta, ove possibile, filtrando sui tipi di tecnologie di ricarica desiderati.

Descrizione dettagliata del problema affrontato

L'applicativo si contestualizza all'interno del trend del momento nel settore dei trasporti: il passaggio ad autoveicoli ad emissioni zero. A riguardo, il mercato è tutt'altro che saturo, presentando un listino inferiore, come numero, a 70 modelli di veicoli diversi (considerando anche alcuni concept che dovrebbero debuttare entro i prossimi 2 anni), di cui la sola marca Tesla occupa una percentuale pari al 20% della scelta totale.

Da qui nasce la volontà di proporre un catalogo completo degli autoveicoli, differenziati non solo per casa produttrice, ma anche per specifiche a livello più tecnico, tali da permettere all'utente una scelta più accurata e, allo stesso tempo, dare vita ad un'analisi di mercato. Analisi di mercato basata su un confronto dei rapporti autonomia/prezzo, ai cui margini si vanno ad inserire indici secondari quali prestazioni, efficienza delle batterie, supporto alla ricarica rapida, presenza della trazione integrale e numero di posti.

A livello di software, per quanto concerne questo primo step, verrà presentato un pannello per l'impostazione, non obbligatoria, di ogni singolo campo di preferenze dell'utente, mediante l'uso di elementi quali menù a tendina (ChoiceBox), caselle di controllo(CheckBox) e slider. Maggiori saranno i filtri adottati, più affinato sarà il risultato della ricerca, con l'unico caso particolare in cui l'utente decidesse di scegliere un modello specifico di autoveicolo: in tal caso verrà mostrata a video tale scelta, corredata nella sua completezza dalle specifiche tecniche.

Al termine del suddetto passaggio, si colloca il vero corpo dell'applicazione, dove sono messe in atto le molteplicità di algoritmi, che verranno descritti in dettaglio nei paragrafi successivi.

Lo scopo di questa seconda parte è di offrire uno strumento di navigazione e di calcolo delle rotte per un possessore di un autoveicolo elettrico, che si troverebbe a dover affrontare, in taluni casi, le problematiche delle soste presso stazioni di rifornimento che prevedono l'esistenza di colonnine di ricarica; non essendo tutto ciò diventato uno standard al giorno d'oggi, neanche in un paese tecnologicamente avanzato come la California (territorio di riferimento in cui si contestualizza questo applicativo), risulta utile la conoscenza a priori della

collocazione dei punti presso cui dover fare tappa. A ciò si aggiunge la possibilità, ove consentito, di applicare un ulteriore filtro circa la visibilità delle sole stazioni con supporto alle tecnologie di ricarica più rapida.

L'utente, quindi, dovrà impostare punti di partenza e arrivo, inclusi tra le centinaia di città e province presenti nello stato californiano, per ottenere tutte le informazioni sul percorso, sia sotto forma di resoconto scritto, sia sotto forma grafica su una finestra di Google Maps.

Le due funzionalità descritte nascono con l'idea di funzionare sequenzialmente: il risultato delle preferenze sull'autoveicolo (che potrà essere eventualmente anche più di uno e in tal caso verrà valutato il migliore secondo il rapporto autonomia/prezzo) va a coniugarsi in un calcolo del percorso basato sul modello stesso.

Per lasciare maggiore dinamicità e libertà all'utente, come è possibile avvalersi, solamente, della scelta dell'auto, così viene reso libero il calcolo del percorso come prima operazione all'avvio: in tal caso il programma userà un modello di autoveicolo casuale tra quelli presenti nel database.

Descrizione del data-set utilizzato per l'analisi.

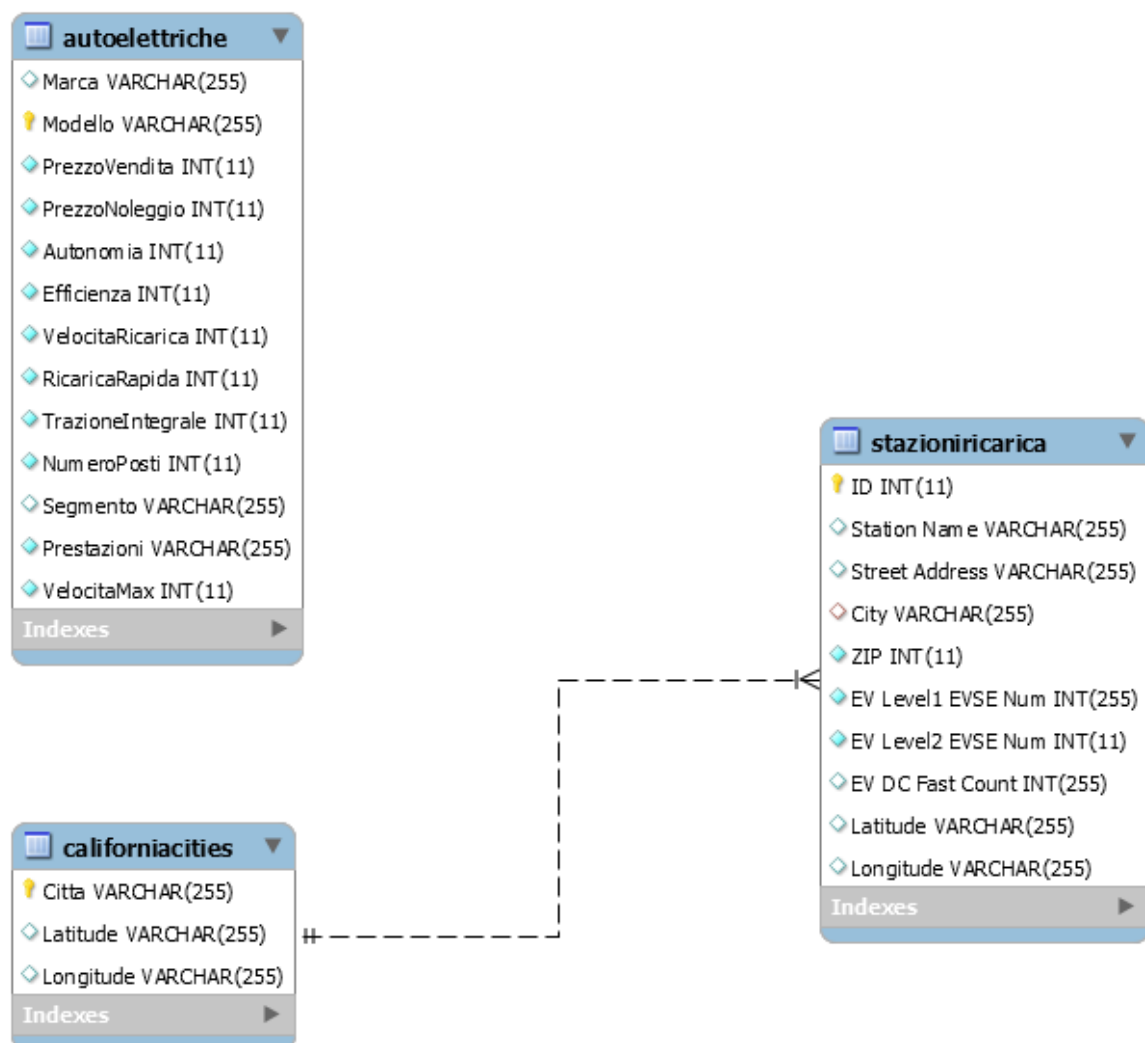
I dati di tale software nascono a seguito di una ricerca accurata, che fosse quanto più aggiornata possibile. I data-set presenti sono tre (lista degli autoveicoli elettrici, elenco delle città e province dello stato della California e mappa delle stazioni di ricarica presenti sul territorio della regione), di cui uno, quello degli autoveicoli elettrici, costruito da me facendo riferimento alle informazioni riportate dalle maggiori piattaforme web del settore, in particolare ev-database.org/ev-database.uk.

Delle tre collezioni, sicuramente, il database più corposo è quello che riguarda la totalità delle stazioni di ricarica. I dati superano le 5000 occorrenze e sono ciascuno identificati da una chiave primaria (voce "ID"), dal nome della stazione di ricarica, dall'indirizzo presso cui si trovano, dalla città dove sono installate e il relativo ZIP code (equivalente del CAP italiano) e da ulteriori informazioni come il numero di colonnine con le relative tecnologie di ricarica supportate (ricarica lenta: voci "EV Level1 EVSE Num" ed "EV Level2 EVSE Num" e veloce: voce "EV DC Fast Count"). Infine, come ultimi due campi, le posizioni geografiche espresse in termine di latitudine e longitudine in gradi decimali. La fonte presso cui è stato trovato tale database è:

https://afdc.energy.gov/fuels/electricity_locations.html#/analyze?region=US-CA&fuel=ELEC&ev_levels=all&show_map=true, una piattaforma federale del governo americano, che dispone di dati open-source accessibili da chiunque. Originariamente tale struttura dati disponeva di più campi, quali numero di telefono del complesso di rifornimento, orari di apertura e chiusura e copie dell'informazioni suddette in lingua francese, che ho deciso di omettere in quanto ritenute ridondanti per lo scopo principe del problema.

Il data-set sulle città e province californiane, invece, è stato reperito da https://www.kaggle.com/camnugent/california-housing-feature-engineering#cal_cities_lat_long.csv, anch'esso contenente dati di natura libera. Consiste di una tabella minimale, dotata di nome della città/provincia e coordinate geografiche sotto forma di latitudine e longitudine in gradi decimali. Per correttezza, da riportare il fatto che originariamente il database aveva informazioni solo su una parte del totale delle città a province (pari circa all'80%, dovuto probabilmente al fatto che certe voci le annetteva sotto macro agglomerati, quali metropoli).

Infine, la tabella sui modelli di autoveicoli, riporta, al giorno d'oggi, 66 modelli differenti di 25 case automobilistiche distinte. Per ciascun modello, oltre al riferimento alla casa e al nome, sono riportate informazioni sull'autonomia, efficienza e velocità di ricarica della batteria, supporto alla ricarica rapida e presenza della trazione integrale, prestazioni da 0 a 100 km/h e velocità massima raggiungibile e infine riguardo a un eventuale prezzo di vendita e noleggio (per i modelli che devono ancora debuttare, sono valori stimati).



Descrizione ad alto livello delle strutture dati e algoritmi utilizzati

L'applicazione è stata realizzata mediante l'uso del linguaggio Java, nella sua declinazione *JavaFX*, volta ad una miglior gestione del codice per il *pattern* MVC (Model View Controller). Come strumento per realizzare la grafica, mi sono avvalso dell'applicativo *SceneBuilder*.

Oltre al sopracitato pattern MVC, sono stati utilizzati anche lo schema del *modello DAO* (Data-Access-Object), per tutto ciò che concerne le interazioni col database, e lo *schema ORM* (Object-Relational-Mapping)

I tre package che distinguono l'applicativo software sono:

Package Controller: contenente le classi *Main*, i tre controller (*PannelloInizialeController*, *PannelloSceltaAutoController* e *PannelloCalcoloPercorsoController*) relativi ai tre file .fxml per la definizione dell'interfaccia grafica.

Package DB: contenente le classi *ConnectDB* (gestione delle connessioni al database) e *ColonnineDAO*, classe mediatrice tra il Model e i Data-set.

Package Model: contenente la classe *TesiModel* (e il relativo *TestModel*), oltre che i vari Java Bean: *City*, *AutoElettriche*, *StazioniRicarica*, *ArcoStazione*, *Vertex*. All'interno di questo package avviene la manipolazione dei dati passati dai vari controller al Model.

L'applicazione all'avvio si presenta con un semplice pannello (*PannelloSceltaIniziale*) il cui unico scopo è reindirizzare l'utente verso una delle due funzionalità principali implementate: la scelta dell'autoveicolo o il calcolo del percorso. Tutto ciò, viene reso possibile mediante l'apertura di stage secondari (e la conseguente chiusura del pannello scatenante) che portano al passaggio alla funzionalità desiderata. Ogni pannello può essere richiamato dagli altri due, ad eccezione del pannello della scelta dell'autoveicolo che, per progettazione, non può essere aperto dal pannello di calcolo del percorso.

Aperto lo stage per la scelta dell'autoveicolo, l'utente si troverà di fronte a un'interfaccia semplice ed intuitiva che permetterà, mediante l'uso di indicatori sotto forma di menù a tendina, checkbox e slider, di impostare dei filtri. Ogni dato caricato, che si tratti di menù a tendina o di slider (limiti massimo e minimo), vengono letti dal relativo database all'avvio del pannello e si

aggiornano dinamicamente man mano che l'utente prosegue con l'applicazione di nuovi filtri. Una dimostrazione è ravvisabile nel momento in cui l'utente scegliesse una marca precisa di auto, che comporterebbe la vista dei soli modelli relativi a quella casa automobilistica o, ancora, la scelta di un modello particolare porterebbe alla disabilitazione degli altri campi di scelta, in quanto risulterebbero di nulla utilità. Per l'output dei risultati della ricerca è stato scelto un formato tabellare, più consono da un punto di vista di ordine grafico. Infine, si cita, la possibilità di resettare tutti i filtri inseriti mediante l'azione sul relativo bottone.

Se invece, fin dall'inizio, l'utente avesse scelto di passare subito al calcolo del percorso per la simulazione, il programma, sceglierebbe attraverso una funzione randomica, uno dei modelli presenti nel data-set (di modo da caratterizzare la successiva costruzione del grafo e i relativi calcoli associati). Anche in questo caso, sarebbe richiesta un'interazione dell'utente per ciò che riguarda l'impostazione dei punti di partenza ed arrivo (a cui seguirebbe una generazione automatica delle relative coordinate in campi di testo non editabili). Non differentemente dall'analogia casistica, i menù a tendina sarebbero popolati all'avvio del pannello mediante una lettura dal data-set 'CaliforniaCities'. Se consentito dal modello di auto scelto (o generato) l'utente potrà aggiungere un ulteriore vincolo per il calcolo del percorso ottimale, riguardante la visibilità delle sole stazioni di ricarica, al cui interno siano presenti anche colonnine a ricarica rapida.

L'output del calcolo del percorso verrà mostrato sia testualmente, all'interno di una Text-Area, sia visivamente in una WebView all'interno dell'applicativo che richiama una pagina web di Google Maps (a seguito verrà descritto come).

Se invece, tale pannello, dovesse essere richiamato a seguito della scelta dell'auto (e qualora la scelta fosse avvenuta), l'unica differenza consisterebbe nell'aver a disposizione, come soggetto del calcolo del percorso, la propria scelta di autoveicolo (che nel caso coincidesse con più autoveicoli, risulterebbe quello col miglior rapporto autonomia/prezzo).

A livello algoritmico, la maggior onerosità computazionale si concentra in questo secondo pannello. Per le caratteristiche del problema, si è optato per la costruzione di un grafo non orientato e pesato (*SimpleWeightedGraph*, libreria *JgraphT*), con pesi pari alle distanze in linea d'aria dei vertici.

Il risultato porta ad una struttura estremamente densa, che nella maggior parte dei casi arriva a contare qualche milione di archi, arrivando molto vicino ad una topologia a maglia completa. Per tale motivo, oltre al vincolo sulla distanza massima degli archi (pari all'autonomia massima dell'autoveicolo scelto) si è voluto implementare una restrizione, mediante l'uso di aree circolari di raggio pari alla metà della distanza tra i punti di partenza ed arrivo e con centro il punto medio tra questi ultimi (vincolo accettabile considerando la densità del grafo e i concetti sulla disuguaglianza triangolare, avendo a che fare con distanze in linea d'aria). Infine per caratterizzare ancora di più il contesto, per auto senza supporto alla ricarica rapida, non sono stati considerati i vertici, e i relativi archi, riguardanti quelle stazioni di ricarica che non offrivano altra tecnologia che non fosse quella veloce. Allo stesso modo, in caso l'utente avesse optato per solo stazioni di ricarica rapida, il grafo sarebbe costruito ignorando tutte quelle stazioni con solo colonnine a ricarica lenta (voce del data-set 'StazioniRicarica' "EV DC Fast Count" pari a 0).

Il calcolo del percorso ottimale è stato conseguito mediante l'applicazione dell'*algoritmo di Dijkstra*, che offriva il miglior compromesso tra dimensione del problema e complessità computazionale. È stata usata la classe *DijkstraShortestPath* offerta dalla libreria JGraphT, che permette di ottenere il cammino minimo, in questo caso, basato sulla distanza percorsa.

È stata inserita anche un'implementazione di *Spanning Tree*, mediante la classe *BreadthFirstIterator* (anch'essa importata dalla libreria JGraphT) per far visualizzare all'utente ulteriori stazioni di ricarica presso cui potrebbe imbattersi nel percorso.

Infine, un ultimo accenno alla visualizzazione del percorso mediante Google Maps. Non potendo accedere alle API della casa di Mountain View, ho manipolato la string query presente nell'URL delle pagine di Google Maps, inserendo, o per lo più concatenando, di volta in volta i vari punti presso cui fare tappa, salvati nella struttura dati uscente dal calcolo del cammino minimo.

Di seguito si riportano alcuni passi fondamentali del codice.

public void creaGrafo: metodo di creazione del grafo, con all'interno tutte le stazioni di ricarica presenti a database. Per ottimizzare i tempi si è optato per un approccio basato sul pattern ORM e DAO, in modo da abbattere la percentuale di calcolo computazionale spesa lato Database.

```
//Creazione grafo per Autoveicoli che supportano RicaricaRapida ma che vogliono anche RicaricaLenta
public void creaGrafo(String partenza, String arrivo, int autonomia) {
    City part=this.cities.get(partenza);
    City arr=this.cities.get(arrivo);

    Double latpartenza=part.getCoords().getLatitude();
    Double latarrivo=arr.getCoords().getLatitude();
    Double longpartenza=part.getCoords().getLongitude();
    Double longarrivo=arr.getCoords().getLongitude();
    //Creazione della fascia
    /*Double min;
    Double max;
    if(latpartenza>latarrivo) {
        max=latpartenza;
        min=latarrivo;
    }
    else {
        min=latpartenza;
        max=latarrivo;
    }*/

    Double latcenter=(latpartenza+latarrivo)/2;
    Double longcenter=(longpartenza+longarrivo)/2;
    Double radius=this.distanzaPunti(part.getCoords(), arr.getCoords())/2;

    Graphs.addAllVertices(this.reteStazioni, dao.getListStazioni(this.stations,this.cities, latcenter, longcenter,radius));
    this.reteStazioni.addVertex(part);
    this.reteStazioni.addVertex(arr);

    for(Vertex s: this.reteStazioni.vertexSet()) {
        if(s instanceof StazioniRicarica)
            for(Vertex t: this.reteStazioni.vertexSet()) {
                if(t instanceof StazioniRicarica)
                    if(!((StazioniRicarica)s.equals(t)) && this.distanzaPunti(((StazioniRicarica)s).getCoords(), ((StazioniRicarica)t).getCoords())<autonomia) {
                        //System.out.println(s.toString()+" "+t.toString());
                        ArcoStazione a=new ArcoStazione(s,t,this.distanzaPunti(((StazioniRicarica)s).getCoords(), ((StazioniRicarica)t).getCoords()));
                        Graphs.addEdge(this.reteStazioni, a.getS1(), a.getS2(), a.getDistance());
                    }
            }
    }

    for(Vertex s:this.reteStazioni.vertexSet())
        if(s instanceof City)
            for(Vertex t:this.reteStazioni.vertexSet()) {
                if(t instanceof City) {
                    if(!((City)s.equals((City)t)) && this.distanzaPunti(((City)s).getCoords(), ((City)t).getCoords())<autonomia) {
                        ArcoStazione a=new ArcoStazione(s,t, this.distanzaPunti(((City)s).getCoords(), ((City)t).getCoords()));
                        Graphs.addEdge(this.reteStazioni, a.getS1(), a.getS2(), a.getDistance());
                    }
                }
                else if(this.distanzaPunti(((City)s).getCoords(), ((StazioniRicarica)t).getCoords())<autonomia) {
                    ArcoStazione a=new ArcoStazione(s,t, this.distanzaPunti(((City)s).getCoords(), ((StazioniRicarica)t).getCoords()));
                    Graphs.addEdge(this.reteStazioni, a.getS1(), a.getS2(), a.getDistance());
                }
            }
    }

    System.out.format("Grafo creato con %d vertici e %d archi", this.reteStazioni.vertexSet().size(),this.reteStazioni.edgeSet().size());
}
```

public void creaGrafoSlow e **public void creaGrafoFast:** le due varianti del metodo sovradescritto, che permettono la creazione di due grafi meno densi. Il primo trova applicazione per tutti quegli autoveicoli che non supportano la ricarica rapida e che, di conseguenza, non avrebbero interesse nell'aver visibilità di tutte quelle stazioni di ricarica al cui interno siano presenti solo colonnine a ricarica rapida. Il secondo viene richiamato nel caso in cui l'utente decidesse di ottenere un percorso che comprenda solo quelle stazioni che possano garantire soste più brevi grazie a tecnologie di ricarica più moderne.

```

//Creazione Grafo per Autoveicoli che supportano solo RicaricaLenta
public void creaGrafoSlow(String partenza, String arrivo, int autonomia) {
    City part=this.cities.get(partenza);
    City arr=this.cities.get(arrivo);

    Double latpartenza=part.getCoords().getLatitude();
    Double latarrivo=arr.getCoords().getLatitude();
    Double longpartenza=part.getCoords().getLongitude();
    Double longarrivo=arr.getCoords().getLongitude();
    //Creazione della fascia
    /*Double min;
    Double max;
    if(latpartenza>latarrivo) {
        max=latpartenza;
        min=latarrivo;
    }
    else {
        min=latpartenza;
        max=latarrivo;
    }*/
    Double latcenter=(latpartenza+latarrivo)/2;
    Double longcenter=(longpartenza+longarrivo)/2;
    Double radius=this.distanzaPunti(part.getCoords(), arr.getCoords())/2;

    Graphs.addAllVertices(this.reteStazioni, dao.getListStazioniSlow(this.stations,this.cities, latcenter, longcenter, radius));
    this.reteStazioni.addVertex(part);
    this.reteStazioni.addVertex(arr);

    for(Vertex s: this.reteStazioni.vertexSet()) {
        if(s instanceof StazioniRicarica)
            for(Vertex t: this.reteStazioni.vertexSet()) {
                if(t instanceof StazioniRicarica)
                    if(!((StazioniRicarica)s).equals((StazioniRicarica)t) && this.distanzaPunti(((StazioniRicarica)s).getCoords(), ((StazioniRicarica)t).getCoords())<autonomia ) {
                        //System.out.println(s.toString()+" "+t.toString());
                        ArcoStazione a=new ArcoStazione(s,t,this.distanzaPunti(((StazioniRicarica)s).getCoords(), ((StazioniRicarica)t).getCoords()));
                        Graphs.addEdge(this.reteStazioni, a.getS1(), a.getS2(), a.getDistance());
                    }
            }
    }

    for(Vertex s:this.reteStazioni.vertexSet())
        if(s instanceof City)
            for(Vertex t:this.reteStazioni.vertexSet()) {
                if(t instanceof City) {
                    if(!((City)s).equals((City)t) && this.distanzaPunti(((City)s).getCoords(), ((City)t).getCoords())<autonomia) {
                        ArcoStazione a=new ArcoStazione(s,t, this.distanzaPunti(((City)s).getCoords(), ((City)t).getCoords()));
                        Graphs.addEdge(this.reteStazioni, a.getS1(), a.getS2(), a.getDistance());
                    }
                }
                else if(this.distanzaPunti(((City)s).getCoords(), ((StazioniRicarica)t).getCoords())<autonomia) {
                    ArcoStazione a=new ArcoStazione(s,t, this.distanzaPunti(((City)s).getCoords(), ((StazioniRicarica)t).getCoords()));
                    Graphs.addEdge(this.reteStazioni, a.getS1(), a.getS2(), a.getDistance());
                }
            }
    }

    System.out.format("Grafo creato con %d vertici e %d archi", this.reteStazioni.vertexSet().size(),this.reteStazioni.edgeSet().size());
}

```

```

public void creaGrafoFast(String partenza, String arrivo, int autonomia) {
    City part=this.cities.get(partenza);
    City arr=this.cities.get(arrivo);

    Double latpartenza=part.getCoords().getLatitude();
    Double latarrivo=arr.getCoords().getLatitude();
    Double longpartenza=part.getCoords().getLongitude();
    Double longarrivo=arr.getCoords().getLongitude();
    //Creazione della fascia
    /*Double min;
    Double max;
    if(latpartenza>latarrivo) {
        max=latpartenza;
        min=latarrivo;
    }
    else {
        min=latpartenza;
        max=latarrivo;
    }*/
    Double latcenter=(latpartenza+latarrivo)/2;
    Double longcenter=(longpartenza+longarrivo)/2;
    Double radius=this.distanzaPunti(part.getCoords(), arr.getCoords())/2;

    Graphs.addAllVertices(this.reteStazioni, dao.getListStazioniFast(this.stations,this.cities, latcenter, longcenter, radius));

    this.reteStazioni.addVertex(part);
    this.reteStazioni.addVertex(arr);

    for(Vertex s: this.reteStazioni.vertexSet()) {
        if(s instanceof StazioniRicarica)
            for(Vertex t: this.reteStazioni.vertexSet()) {
                if(t instanceof StazioniRicarica)
                    if(!((StazioniRicarica)s).equals((StazioniRicarica)t) && this.distanzaPunti(((StazioniRicarica)s).getCoords(), ((StazioniRicarica)t).getCoords())<autonomia ) {
                        //System.out.println(s.toString()+" "+t.toString());
                        ArcoStazione a=new ArcoStazione(s,t,this.distanzaPunti(((StazioniRicarica)s).getCoords(), ((StazioniRicarica)t).getCoords()));
                        Graphs.addEdge(this.reteStazioni, a.getS1(), a.getS2(), a.getDistance());
                    }
            }
    }

    for(Vertex s:this.reteStazioni.vertexSet())
        if(s instanceof City)
            for(Vertex t:this.reteStazioni.vertexSet()) {
                if(t instanceof City) {
                    if(!((City)s).equals((City)t) && this.distanzaPunti(((City)s).getCoords(), ((City)t).getCoords())<autonomia) {
                        ArcoStazione a=new ArcoStazione(s,t, this.distanzaPunti(((City)s).getCoords(), ((City)t).getCoords()));
                        Graphs.addEdge(this.reteStazioni, a.getS1(), a.getS2(), a.getDistance());
                    }
                }
                else if(this.distanzaPunti(((City)s).getCoords(), ((StazioniRicarica)t).getCoords())<autonomia) {
                    ArcoStazione a=new ArcoStazione(s,t, this.distanzaPunti(((City)s).getCoords(), ((StazioniRicarica)t).getCoords()));
                    Graphs.addEdge(this.reteStazioni, a.getS1(), a.getS2(), a.getDistance());
                }
            }
    }

    System.out.format("Grafo creato con %d vertici e %d archi", this.reteStazioni.vertexSet().size(),this.reteStazioni.edgeSet().size());
}

```

public List<Vertex> verticiRaggiungibili, spanningTreeFinoA, calcoloCamminoMinimo: rappresentano i tre algoritmi che operano sul grafo creato e il cui scopo è quello di fornire informazioni circa le stazioni di ricarica raggiungibili (i primi due, cooperano) e il percorso ottimo, sfruttando concetti di Teoria sui Grafi quali: Spanning Tree (Albero Ricoprente), Cammino Minimo (attraverso l'applicazione dell'Algoritmo di Dijkstra).

```
public List<Vertex> verticiRaggiungibili(Vertex source){
    backVisit=new HashMap<>();
    List<Vertex> result=new ArrayList<>();

    GraphIterator<Vertex, DefaultWeightedEdge> it= new BreadthFirstIterator<>(this.reteStazioni, source);
    it.addTraversallListener(new TesiModel.EdgeTraversedGraphListener());

    backVisit.put(source, null);

    while(it.hasNext()) {
        result.add(it.next());
    }

    return result;
}

public List<Vertex> spanningTreeFinoA(Vertex target){
    List<Vertex> percorso=new LinkedList<>();

    if(!backVisit.containsKey(target)) {
        return null;
    }

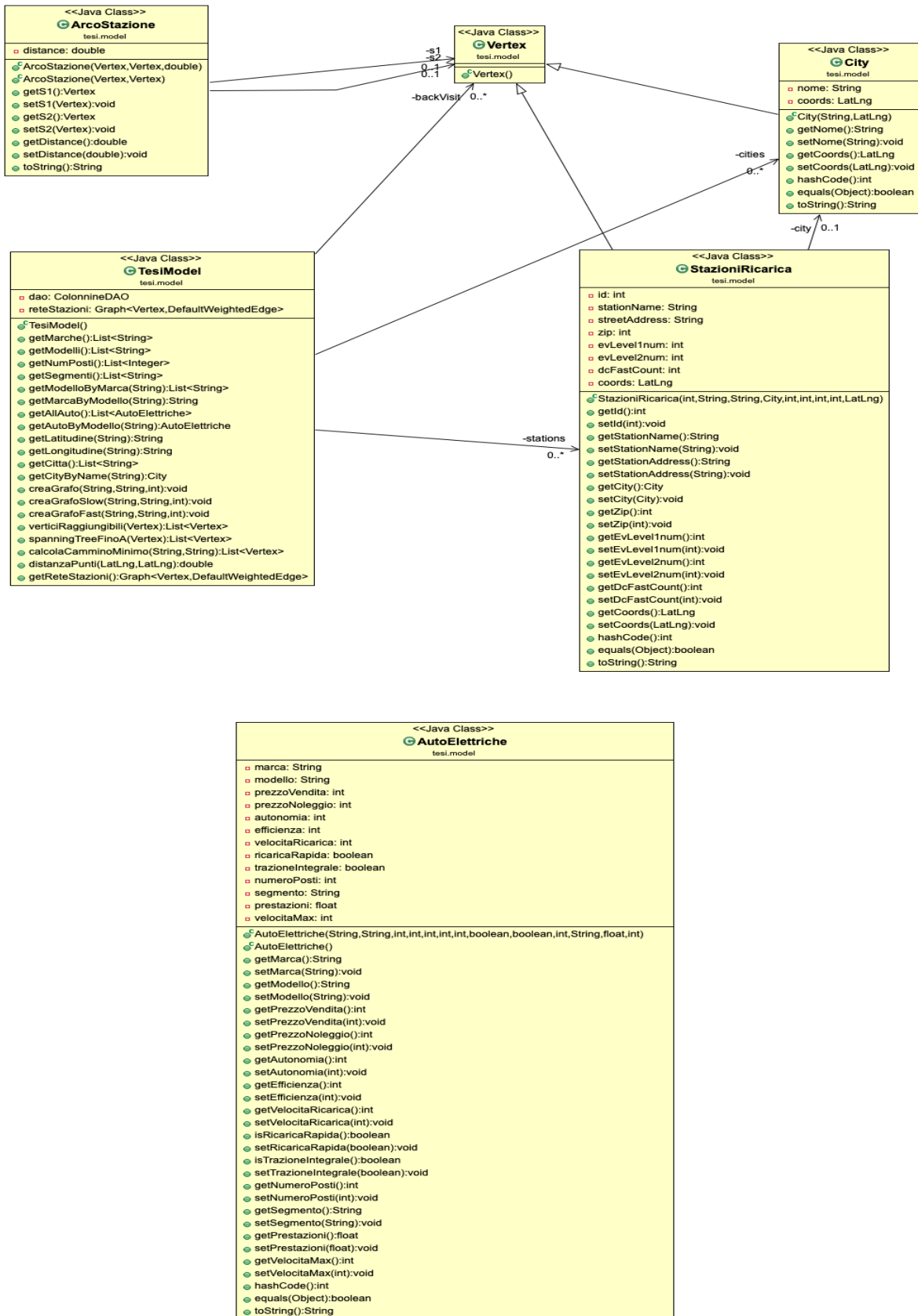
    Vertex v=target;

    while(v!=null) {
        percorso.add(0,v);
        v=backVisit.get(v);
    }

    return percorso;
}

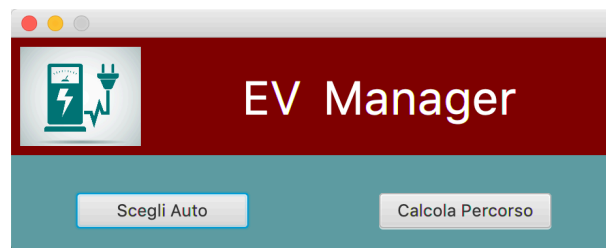
public List<Vertex> calcolaCamminoMinimo(String partenza, String arrivo){
    City source=this.cities.get(partenza);
    City target=this.cities.get(arrivo);
    DijkstraShortestPath<Vertex, DefaultWeightedEdge> dijkstra=new DijkstraShortestPath<>(this.reteStazioni);
    GraphPath<Vertex, DefaultWeightedEdge> path=dijkstra.getPath(source, target);
    return path.getVertexList();
}
```

Diagramma delle classi delle parti principali dell'applicazione



Alcune videate dell'applicazione realizzata e link al video dimostrativo del software


Pannello iniziale all'avvio dell'applicazione



Pannello per la scelta dell'autoveicolo

The screenshot shows the "EV Manager - Scelta Auto" panel. It has a dark red header with a home icon on the left, the text "EV Manager - Scelta Auto" in white, and a right arrow icon on the right. Below the header, there are two rows of filters. The first row includes checkboxes for "Compro" and "Noleggio", followed by dropdown menus for "Marca", "Modello", "Segmento", and "N° Posti", and checkboxes for "Trazione Integrale" and "Ricarica Rapida". The second row includes sliders for "Fascia di Prezzo", "Prezzo Noleggio", "Autonomia", "Velocità Max", and "Prestazioni 0-100". Below the filters, there is a table with columns: "Marca", "Modello", "Prezzo Vendita (€)", and "Prezzo Noleggio". On the left side of the table, there are two buttons: "Risultati" and "Reset".

Pannello per il calcolo del percorso



EV Manager - Calcolo del Percorso

Modello Scelto Kona Electric 39 kWh Autonomia Massima 250

Scegli la Città di Partenza

Partenza Lat Long

Scegli la Città di Arrivo

Arrivo Lat Long


Calcolo del Percorso

☐ Con Colonnine DC Fast

Calcola Percorso

Reset

California, Stati Uniti



California

Stati Uniti

Indicazioni stradali


Salva

Nelle vicinanze

Invia al telefono

Condividi

Foto



Foto

In breve

La California, uno stato occidentale degli Stati Uniti, si estende per quasi 900 miglia lungo la costa del Pacifico partendo dal confine con il Messico. Il suo territorio comprende spiagge con scogliere, foreste di sequoie, le Montagne della Sierra Nevada, i terreni agricoli della Central Valley e il deserto del Mojave. La città di Los Angeles è sede dell'industria dell'intrattenimento di Hollywood. San Francisco, che si sviluppa su colline, è conosciuta per il Golden Gate Bridge.

Promemoria sulla privacy di Google

RICORDAMELO PIÙ TARDI

LEGGI

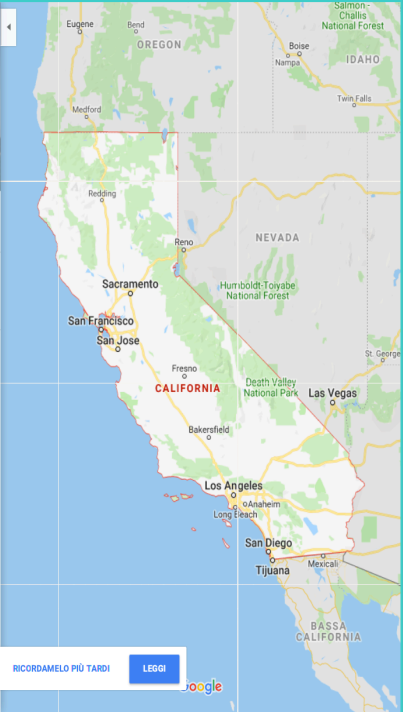


Tabelle con risultati sperimentali ottenuti

L'esempio mostrato riguarda la scelta tra gli autoveicoli di un noto marchio tedesco. Come ulteriori filtri si è voluto impostare il segmento, per la scelta di un Suv, la presenza della trazione integrale e della ricarica rapida e delle soglie accettabili come prezzo di vendita, autonomia della batteria, picchi di velocità e prestazioni.

EV Manager - Scelta Auto

☒ Compro Marca Modello Segmento N° Posti ☒ Trazione Integrale

☐ Noleggio Audi Suv ☒ Ricarica Rapida

Fascia di Prezzo Prezzo Noleggio Autonomia Velocità Max Prestazioni 0-100

0 105.000 210.000 0 800 1.600 2.400 0 275 550 825 120 220 320 410 0 4 8 12 16 20

Risultati

Marca	Modello	Prezzo Vendita (€)	Prezzo Noleggio
Audi	e-tron 55 quattro	84100	850
Audi	e-tron Sportback	85000	850
Audi	Q4 e-tron	60000	600

Reset

Modello Scelto

Q4 e-tron

Autonomia Massima

425

Scegli la Città di Partenza

Partenza

San Francisco

Lat

37.774932

Long

-122.419418

Scegli la Città di Arrivo

Arrivo

Los Angeles

Lat

34.052234

Long

-118.2436819

Calcolo del Percorso

Con Colonnine DC Fast

Calcola Percorso

Reset

Distanza tra i due punti: 559.1232746876584 km

Possibili stazioni di ricarica presenti nel percorso tra San Francisco e Los Angeles

- Chargepoint: 3433 ROBERTO CT, SAN LUIS OBISPO

Il percorso più veloce per raggiungere Los Angeles da San Francisco è:

- San Francisco
- Chargepoint: 2301 TECHNOLOGY PKWY, HOLLISTER
- Chargepoint: 6514 LANKERSHIM BLVD, NORTH HOLLYWOOD
- Los Angeles

San Francisco, California, Stati Uniti

2301 Technology Pkwy, Hollister, CA 95

6514 Lankershim Blvd, North Hollywood

Los Angeles, California, Stati Uniti

Aggiungi destinazione

OPZIONI

Invia indicazioni stradali al tuo telefono

tramite US-101 S

6 h 19 min

6 h 19 min senza traffico

397 miglia

DETTAGLI

Mendocino National Forest

Reno

Carson City

Eldorado National Forest

Stanislaus National Forest

Yosemite National Park

Sierra National Forest

Sequoia National Forest

Death Nat

San Francisco

Oakland

San Jose

2301 Technology Parkway

Monterey

Visalia

Fresno

California

San Luis Obispo

Santa Maria

Los Padres National Forest

Santa Barbara

6514 Lankershim Boulevard

Los Angeles

Long Beach

Valutazioni sui risultati ottenuti e conclusioni.

I tempi di esecuzione dell'applicativo, nonostante l'enorme mole di dati, sono modesti. La maggior parte del calcolo computazionale, si concentra nel calcolo del percorso e i fattori che incidono di più nelle tempistiche sono variabili. Il caso peggiore andrebbe a verificarsi con la contemporaneità di un autoveicolo dall'autonomia elevata (generalmente sopra i 500-600 km di autonomia), di due punti di partenza ed arrivo ai due estremi dello stato californiano e dalla scelta di un grafo che comprenda sia colonnine a ricarica lenta che a ricarica veloce. In tal caso il grafo, si stima, nell'ordine delle decine di milioni di archi con un numero di vertici quasi pari al totale dei vertici possibili (5061 vertici al massimo).

La forza di questo applicativo, si ravvisa nella semplicità d'uso e nell'ottimizzazione algoritmica di istruzioni operanti su quantitativi di dati non banali. Inoltre rappresenta un tool di estrema attualità, che riesce ad avvicinare ad un mondo, sì in forte crescita, su cui però c'è ancora scetticismo, ignoranza e disinformazione.

Ogni dato che viene mostrato all'utente rappresenta un'informazione o una stima reale; le poche semplificazioni apportate sono state scelte obbligate per la capacità del programmatore e il limitato hardware presente dietro questo applicativo.

Un modo per rendere ancora più snella, lato client, l'applicazione, sarebbe quello di poter fornire un grafo di base già creato all'avvio dell'applicativo, dal momento che va ad occupare il 90-95% delle tempistiche totali di esecuzione.

Link al video YouTube: <https://youtu.be/Xsyuer7aVWo>



Quest'opera è distribuita con licenza Creative Commons Attribuzione - Non commerciale - Condividi allo stesso modo 4.0 Internazionale.

Copia della licenza consultabile al sito web: <https://creativecommons.org/licenses/by-nc-sa/4.0/>