Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica per il management

RIORGANIZZAZIONE DI ROTTE PRESTABILITE IN BASE AD AREE PER LA DISTRIBUZIONE URBANA DI MERCI: ALGORITMO ED APPLICAZIONE MOBILE

Relatore:
Chiar.mo Prof.
LUCIANO BONONI
Correlatore:
Dott.
LUCA BEDOGNI

Presentata da:
VIVIANA RAFFA

Sessione II
Anno Accademico 2017-2018

Introduzione

Negli ultimi tempi si sente sempre più spesso parlare di città che decidono di iniziare a piccoli passi la trasformazione in Smart City, ovvero "città intelligente". Questo perché la gestione delle aree urbane è diventata sempre più complicata con l'aumentare della concentrazione della popolazione; si cerca quindi di trovare soluzioni, basandosi anche su tecnologie innovative, che permettano di affiancare le amministrazioni nella governance delle pratiche della città e di migliorare la qualità della vita dei cittadini.
Uno dei servizi al momento più sviluppati nel mondo "smart" è la mobilità, con la quale è possibile influire sul fronte economico ed ecologico. La Smart Mobility infatti, comprende azioni per diminuire l'inquinamento e gli sprechi cercando, allo stesso tempo, di aumentare l'efficienza del trasporto riducendo tempi e costi. Questo settore è in forte e continua crescita e le frequenti innovazioni legate agli Intelligent Transportation System (ITS) sono spesso fonte di ispirazione, oltre che di studio, per cercare di sviluppare delle migliorie in questo ambito.
Questo lavoro, dapprima analizza le soluzioni per una "mobilità intelligente" presenti sul mercato, nell'ambito delle app per la ricerca del parcheggio o per calcolare la rotta migliore dato un insieme di punti; in un secondo momento, in base alle necessità riscontrate, illustra il funzionamento e l'evoluzione di un algoritmo che, preso un insieme di punti geografici in input, restituisce un elenco di parcheggi scelti in base alla vicinanza con i punti dell'itinerario. Infine, propone un'applicazione mobile, che supporti ed integri quelle già esistenti, per corrieri di merci a Barcellona, suggerendo all'utente dove par-

cheggiare con il veicolo in maniera tale da ottimizzare i tempi di percorrenza della rotta ed evitare emissioni inquinanti nelle zone abitate. In particolare, i parcheggi vengono scelti tra l'insieme delle aree DUM (Distribuzione Urbana di Merci), istituite da alcuni anni nella capitale della Catalogna, ovvero zone di sosta riservate esclusivamente al carico e scarico di merci, nell'ottica di facilitare i corrieri nelle consegne e di ridurre i disagi alla mobilità urbana.
Nel Cap. 1 vengono introdotti i concetti di Smart City e Smart Mobility, e viene presentata la problematica del calcolo delle rotte con alcuni esempi di applicazioni pratiche. Nel Cap. 2 si descrive lo scenario da cui ha avuto origine questo progetto: le peculiarità della città di Barcellona e il motivo del ricorso alle aree DUM. Nel Cap. 3 si procede illustrando l'evoluzione del processo di progettazione dell'algoritmo che permette la riorganizzazione delle rotte e le strutture dati utilizzate come supporto all'applicazione. Infine, nel Cap. 4 viene illustrato il funzionamento e la struttura interna dell'applicazione mobile Android che mette in pratica l'algoritmo.

Indice

Elenco delle figure

Capitolo 1

Stato dell'arte

1.1 Smart City

Molte città, di grandi e medie dimensioni, si trovano ad affrontare difficoltà nella gestione e nel miglioramento delle loro aree urbane. L'impatto sulla vita quotidiana diventa sempre più evidente con l'aumento della concentrazione della popolazione. Negli ultimi anni sono stati osservati grandi cambiamenti nell'ambiente con un significativo impatto ecologico, sociale e finanziario. Inoltre, man mano che le città crescono, richiedono maggiori contributi finanziari per mantenere le loro infrastrutture urbane.
In questo contesto, una delle principali sfide della società è trasformare le attuali città, o anche costruirne di nuove, con particolare attenzione alla riduzione dell'inquinamento, all'accesso equo alle risorse essenziali (come acqua, elettricità e comunicazioni) ed alle "urban networks" che siano in grado di sopportare i molteplici spostamenti quotidiani. Questi aspetti sono fortemente legati al concetto di città intelligente, che si basa su modi innovativi per gestire la città, compresa una governance partecipativa con i cittadini, le amministrazioni e l'industria. L'obiettivo è proporre servizi (come mobilità, economia collaborativa meglio conosciuta come "sharing economy", aree urbane ecologiche ed inclusione dei cittadini) in grado di far fronte a problemi sociali, economici ed ecologici.[1]

1.2 Smart Mobility

La Smart Mobility costituisce una componente ricorrente nei discorsi sulle Smart City e rappresenta uno dei sei assi su cui è strutturato il concetto di Smart City nella fondativa ricerca condotta dalle università di Vienna, Delft e Lubiana (Giffinger et al. 2007). Rientra, inoltre, tra gli ambiti principali dell'iniziativa "Smart cities and communities" lanciata dall'Unione europea nel 2011, ed è uno dei settori finanziati nei bandi connessi; tra le sedici aree tematiche del programma "Smart cities and communities and social innovation", promosso dal MIUR nel 2012, due sono rappresentate da "Smart mobility" e "Last mile logistic".
Alla base di questa centralità vi sono almeno due fattori. In primo luogo, la mobilità gioca un ruolo molto significativo nei consumi energetici, che costituiscono la dimensione ambientale principale, se non in alcuni casi quasi esclusiva. In secondo luogo, da ormai trent'anni il settore dei trasporti è già ambito privilegiato d'applicazione di innovazioni legate alle ICT, volte a sviluppare i cosiddetti Intelligent Transportation Systems (ITS); i risultati finora raggiunti dagli ITS restano però in gran parte frammentari, eterogenei e poco interoperabili, al punto che la Commissione europea ha adottato nel 2008 un Piano d'azione e nel 2010 una Direttiva con l'obiettivo principale di armonizzare, su standard comuni, gli ITS operanti in tutto il territorio dell'Unione. E proprio perché poco "maturo", il mercato di questi sistemi è estremamente interessante per le grandi compagnie delle ICT.
Nella smart mobility confluiscono due aspetti: la mobilità sostenibile e mezzi di trasporto tecnologicamente avanzati, il tutto "customizzato" sulle esigenze del cittadino. Comprende azioni per ridurre l'inquinamento e gli sprechi ma allo stesso tempo aumentare l'efficienza del trasporto; si basa sulla creazione di economie di scala per gli spostamenti di persone e merci, mira a migliorare la logistica con l'utilizzo della tecnologia e quindi ci permette di risparmiare tempo e costi. Una mobilità moderna ed efficiente significa spostamenti facili e quindi una qualità di vita nettamente migliore dell'attuale. Allo stesso tempo, spinge ad una svolta ecologica delle persone e dei mezzi (si arriva

prima e si inquina meno).
La smart mobility, oltre a preoccuparsi di abbattere sprechi, emissioni nocive ed, in generale, dell'impatto ambientale dei trasporti, è anche finalizzata a fornire servizi utili per il cittadino, ciò avviene con il fondamentale supporto della tecnologia: con essa si possono, ad esempio, pianificare i viaggi, ottimizzare le rotte, creare sistemi per pagamenti/prenotazioni in mobilità o di "information rescue" attraverso Car-to-Car (C2C) e Car-to-Infrastructure (C2I) Communications.
La smart mobility presenta quindi varie potenzialità nel perseguimento di una maggiore sostenibilità dei sistemi di trasporto, in termini economici (ad esempio, fluidificazione del traffico e dei livelli di congestione) ed ambientali (riduzione dei consumi e delle emissioni). L'attualizzazione di queste potenzialità, e l'entità degli impatti positivi che esse determineranno, dipende però da come le tecnologie, su cui la smart mobility è incentrata, vengono incorporate dagli utenti nelle proprie pratiche ed attività quotidiane. Non si può infatti dare per scontato che queste tecnologie, e in particolare la maggiore quantità e qualità delle informazioni che esse garantiscono, si traducano automaticamente in comportamenti di mobilità più sostenibili. [2]

1.3 Calcolo di rotte (quasi) ottimizzate

Per avere un esempio di possibile applicazione di smart mobility non si deve guardare molto lontano, basti pensare all'uso quasi quotidiano che viene fatto dei servizi per l'instradamento dei veicoli o per avere informazioni in tempo reale sul traffico, sulla disponibilità di mezzi di trasporto a noleggio (auto, biciclette), ecc..
In particolare, il servizio di navigazione da un punto di partenza A ad un punto di destinazione B, oltre ad essere utilizzato dai privati, viene spesso utilizzato dai corrieri incaricati di consegnare merci in diversi punti della città. I cosiddetti "drivers" hanno una reale esigenza di conoscere in percorso più veloce, in quanto spesso devono consegnare un grande numero di colli in

poco tempo.
Si potrebbe quindi pensare che il corriere necessiti semplicemente di uno dei servizi più in voga come Google Maps o Waze per avere le indicazioni dettagliate che gli permettano di arrivare a destinazione; ma c'è una differenza, l'obiettivo del corriere è quello di pianificare una rotta efficiente con destinazioni multiple.
Immaginando di dover visitare dai 50 ai 200 punti in un giorno, Google Maps o Waze non sono in grado di elaborare la rotta migliore per tutti questi, calcoleranno semplicemente quella migliore in base all'ordine in cui sono stati inseriti i punti. La domanda che ci si pone è quindi "Qual è il percorso più corto possibile che mi permette di visitare tutti questi punti una ed una sola volta?", che rende il problema un classico esempio di applicazione del problema del commesso viaggiatore o travelling salesman problem.
Bisognerà assicurarsi che il personale della compagnia stia effettuando il più alto numero di visite o consegne possibile, nel più breve tempo possibile; si avrà quindi bisogno di un'applicazione che semplifichi l'intero processo di consegna e fornisca loro i percorsi più veloci. Alcune importanti features da considerare importanti sono:

- Integrazione: un'app di route planning che si integri con i più diffusi sistemi GPS come Waze o Google Maps, in modo tale da poter pianificare il tempo di guida e la distanza, anche in tempo reale. Un'ulteriore caratteristica utile sarebbe l'integrazione con i sistemi ERP esistenti (Enterprise resource planning o di pianificazione delle risorse d'impresa), software di gestione che integrano tutti i processi di business rilevanti di un'azienda (vendite, acquisti, gestione magazzino, contabilità ecc.) così da evitare, ad esempio, upload manuali delle informazioni sui clienti.

- Monitoraggio: per tenere traccia di tutte le attività intraprese dallo staff, utilizzando geo-tagging e time-stamping (aggiunta di metadati di identificazione geografica e temporale ai file multimediali) in modo da ottenere la piena visibilità e quindi il controllo delle operazioni.

- Soddisfazione del cliente: l'utilizzo di un pianificatore di percorsi consente di comunicare ai clienti quando è in arrivo una consegna, e, se collegato a un'applicazione di e-commerce, è possibile fornire una prova di consegna elettronica.

- Aumento dei profitti: pianificando i percorsi in modo efficiente, il personale può muoversi più rapidamente, effettuare le consegne in modo più accurato, con conseguente riduzione dei costi per tratta e aumento del margine di profitto per autista/percorso.

- Funzionalità offline: con lo staff che viaggia, non è sempre possibile garantire una buona connessione, è quindi necessaria un'applicazione che funzioni offline in modo che si possa continuare ad utilizzarla ovunque. [3]

1.4 Esempi di applicazioni mobili Android per la pianificazione di rotte

1.4.1 L'applicazione mobile RoadWarrior Route Planner

Con l'applicazione RoadWarrior Route Planner si inseriscono le fermate del percorso attraverso l'app o caricando un foglio di calcolo, si personalizzano i percorsi con le preferenze (es. viaggio di sola andata o evitare i pedaggi) e RoadWarrior restituisce il percorso (Figura 1.1). [12] Le features principali di questa applicazione sono quindi:

- Pianificare percorsi con più destinazioni (8 fermate nella versione base)

- Ottimizzare il percorso in base al minor tempo o alla distanza più breve

- Navigare utilizzando Google Maps, Waze ecc.

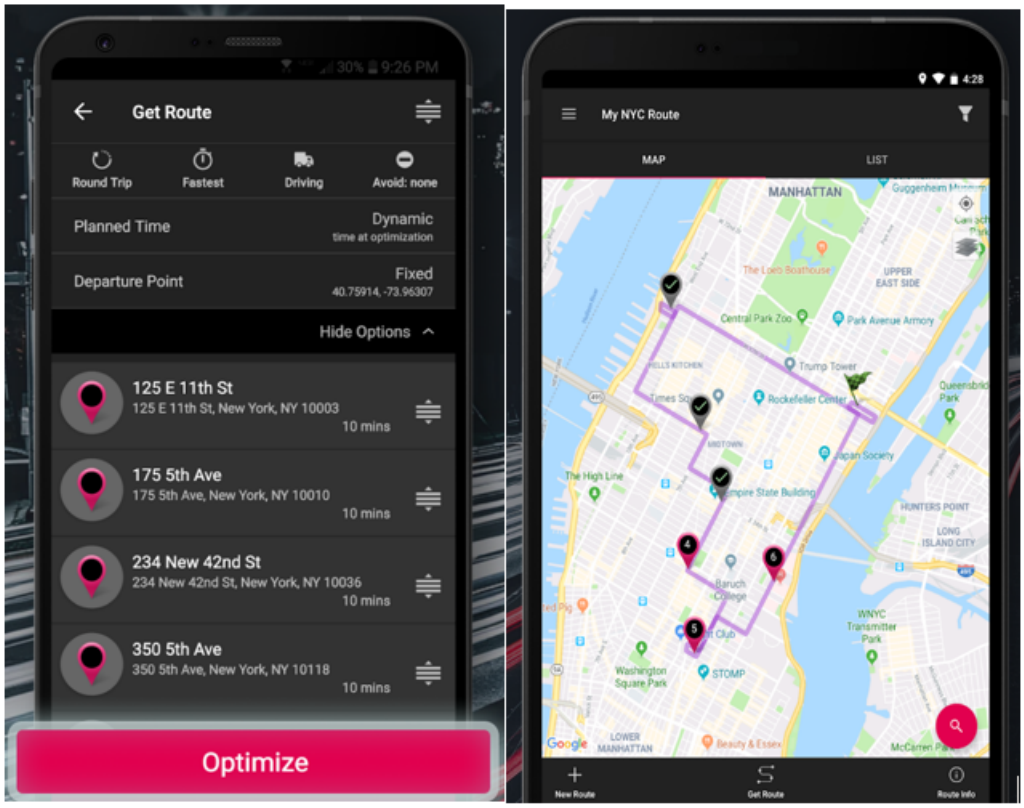

Figura 1.1: Schermate dell'app RoadWarrior Route Planner

- Riordinare manualmente le soste per ricalcolare il percorso in pochi secondi
- Aggiungere finestre temporali per essere sicuro di arrivare durante le ore di apertura o di consegna
- Tenere traccia dei progressi con i check-in mentre si completano le soste
- Inviare ETA (Estimated time of arrival - tempo stimato d'arrivo) e comunicare con i clienti con l'integrazione di Glympse (applicazione per condividere la posizione in tempo reale tramite localizzazione GPS).

La quasi totalità delle applicazioni mobili sono similari, come funzionalità, a RoadWarrior Route Planner.

1.4.2 La piattaforma Cigo!fleet

Cigo!fleet è una piattaforma, composta da una web app ed una mobile app, progettata e sviluppata da Sparsity Technologies; ciò che differenzia Cigo!fleet dalla tipologia di applicazioni presentate in precedenza è essenzialmente il fatto che le rotte vengano calcolate in anticipo, tutte insieme, in un'unica fase e successivamente assegnate ai singoli corrieri; nei precedenti casi veniva dato un insieme di fermate da effettuare ad ogni elemento dello staff e questo aveva la possibilità di creare la rotta inserendo i punti nell'applicazione che preferiva.
Cigo!fleet permette di risparmiare tempo automatizzando il processo di creazione delle rotte, da assegnare all'insieme di corrieri, attraverso un algoritmo creato appositamente ed eseguito a partire dall'insieme dei punti di consegna, importati da un file, che tiene conto di ulteriori variabili come il numero di vettori o la distanza massima che questi possono coprire. Dopo la creazione attraverso un computer, le rotte vengono automaticamente inviate al singolo corriere, che le può controllare accedendo all'applicazione mobile con il suo identificativo (Figura 1.2). Inoltre, il sistema connesso alla rete (Figura 1.3),

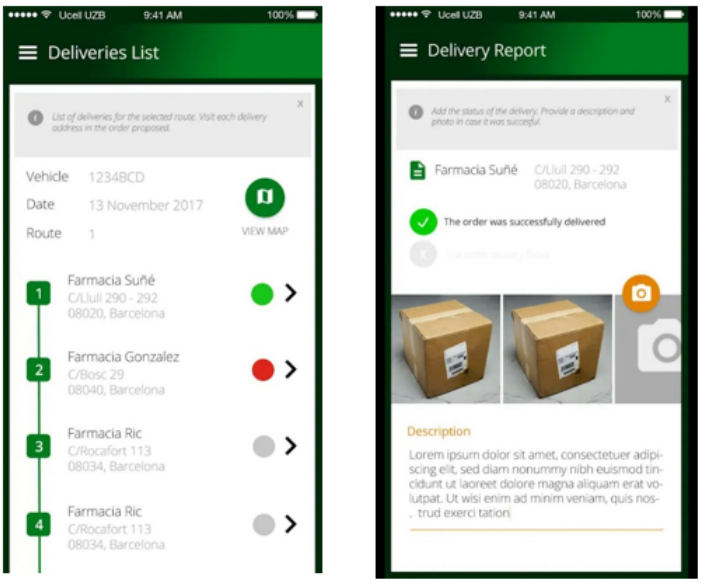

Figura 1.2: Schermate dell'app Cigo!fleet

consente di controllare in tempo reale lo stato delle spedizioni, eventuali ritardi o altri generi di problemi, e di conseguenza, informare il cliente. [11]

1.5 Il problema della ricerca del parcheggio

Ricerche riportano che una percentuale importante del traffico, e quindi dell'inquinamento, nei centri cittadini è dovuta alle auto che girano in cerca di uno spazio per parcheggiare.
Secondo uno studio condotto da Xerox in 19 città europee, gli spagnoli perdono in media 96 ore per trovare un posto per la propria auto, una media di 15 minuti al giorno e un totale di 4 giorni all'anno. Questo tempo di attesa si moltiplica in città dove il traffico è molto più denso, come Londra o Parigi, dove i conducenti di solito hanno bisogno di una media di 20-30 minuti per parcheggiare. Le percentuali degli studi su altre città in Europa e Stati Uniti variano sensibilmente - dal 10 fino al 70 per cento - ma, numeri a parte, l'esperienza quotidiana di molti conferma che il parcheggio è un problema che

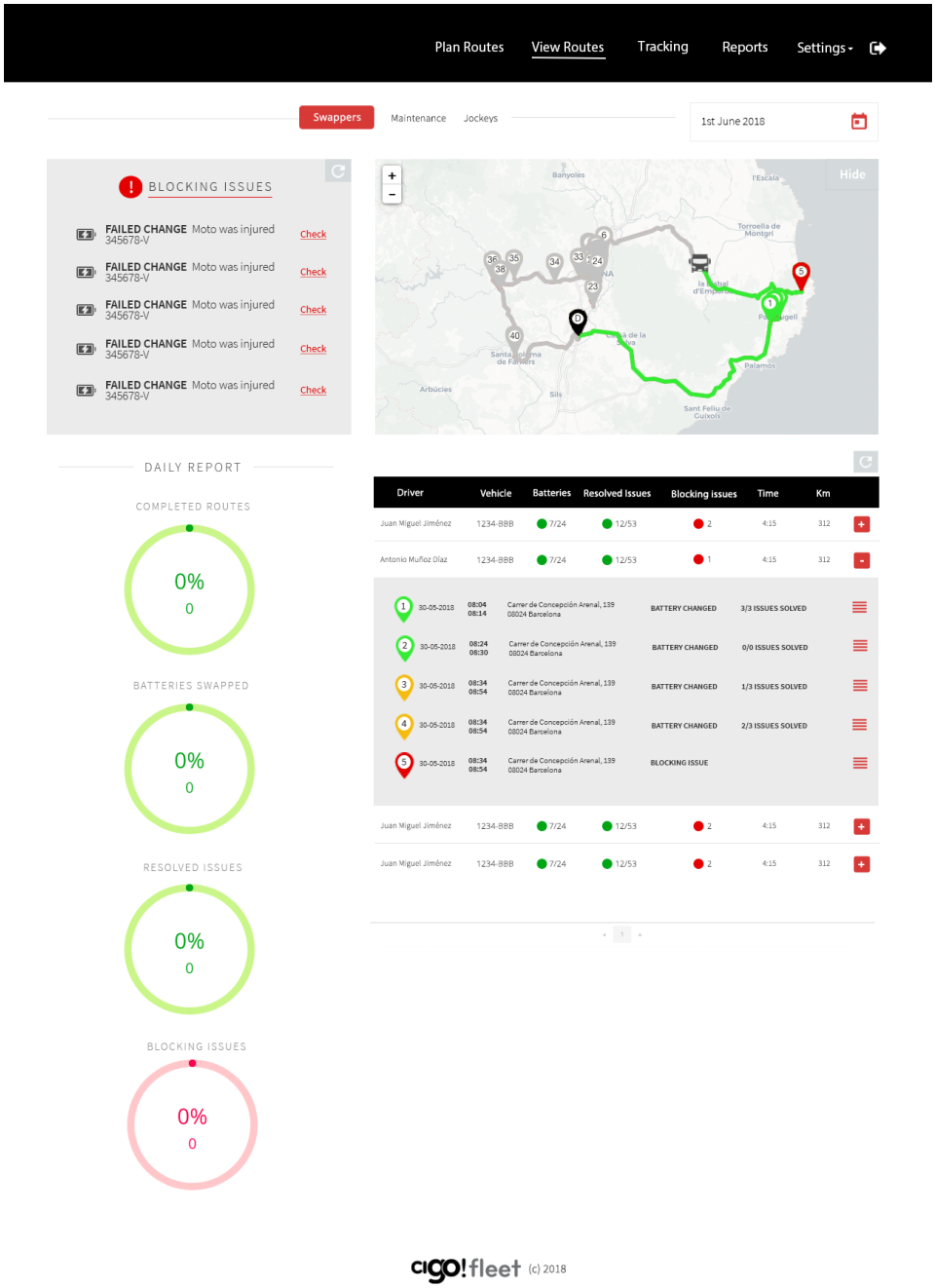

Figura 1.3: Il controllo delle rotte assegnate ai corrieri attraverso la piattaforma aziendale di Cigo!fleet

costa tempo e risorse al singolo e alla collettività.
I dati possono aiutare in questo: le città possono usare dati di occupazione in tempo reale e migliori sistemi di pagamento per offrire un parcheggio più semplice ed efficiente per i conducenti. Grazie a migliori sistemi di tracciamento, informazioni e guida migliorate, le città possono gestire attivamente la congestione relativa ai parcheggi, massimizzare le entrate e utilizzare la politiche di prezzo basate sulla domanda per influenzare il comportamento dei conducenti. [4][5][6]

1.6 Esempi di applicazioni mobili Android per la ricerca del parcheggio

1.6.1 L'applicazione Parkopedia Parcheggi

Si tratta di una vera e propria enciclopedia dei parcheggi, con un database di informazioni molto vasto. Si può effettuare una ricerca selezionando diversi criteri: dalle recensioni degli altri utenti, alla distanza, al costo; i prezzi sono attendibili perché spesso segnalati dai viaggiatori, per ogni zona si trova una mappa fornita da Street View e delle foto scattate dagli utenti. Tra le diverse informazioni si possono inoltre visualizzare i servizi, l'autolavaggio nelle vicinanze, la video sorveglianza, l'illuminazione e la sicurezza per ogni singolo parcheggio (Figura 1.4). [10]

1.6.2 L'applicazione Parclick - Trova e prenota parcheggi

Parclick permette di cercare e trovare parcheggio per auto, moto e furgoni in più di 170 città in Italia e in Europa:

- Italia: Roma, Milano, Venezia, Firenze, Pisa e Napoli
- Spagna: Barcellona, Madrid, Valencia, Siviglia, Bilbao e molte altre ancora.

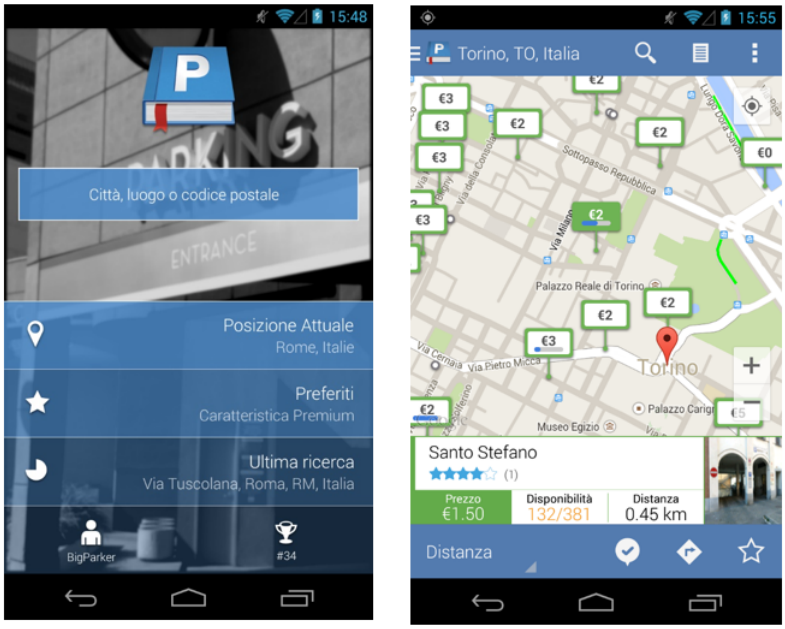

Figura 1.4: Schermate dell'app Parkopedia Parcheggi

- Francia: Parigi, Lione, Nizza, Bordeaux, Reims e Cannes.

- Portogallo: Porto, Lisbona o Coimbra

- Altre città europee: Friburgo, Bruges, Bruxelles, Amsterdam, Ginevra e Basilea, fra le altre

Il funzionamento di Parclick è il seguente: ci si geolocalizza o si cerca una destinazione nell'app o da un qualunque browser web (Figura 1.5) anche su computer, si sceglie tra i parcheggi con le offerte disponibili in zona e si prenota la sosta direttamente dall'app (Figura 1.6); per accedere e parcheggiare il veicolo (macchina, moto, monovolume, furgone, ecc.) per le ore o i giorni di cui si ha bisogno, serve solo il codice di prenotazione. [8]

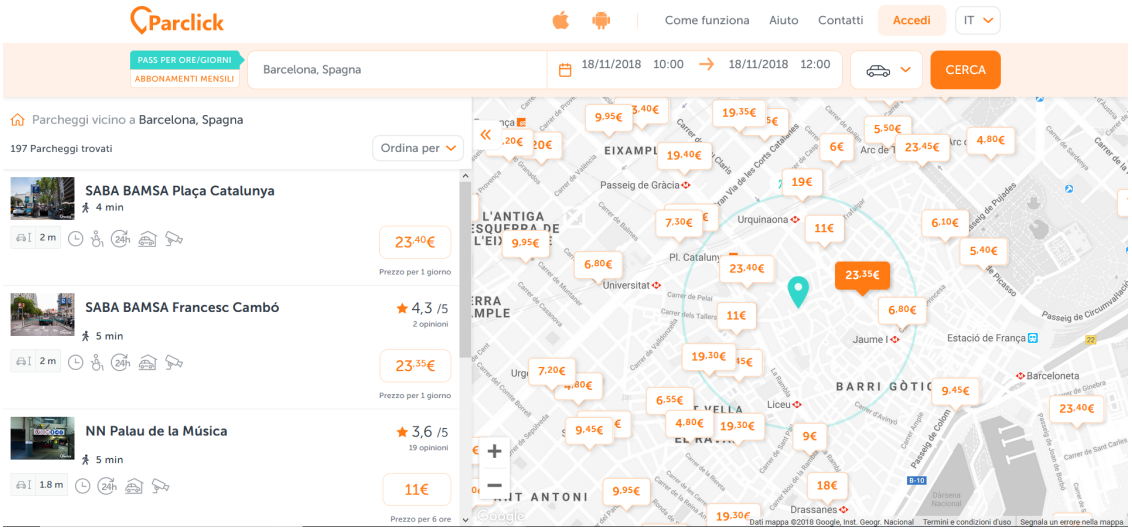

Figura 1.5: Interfaccia della piattaforma Parclick

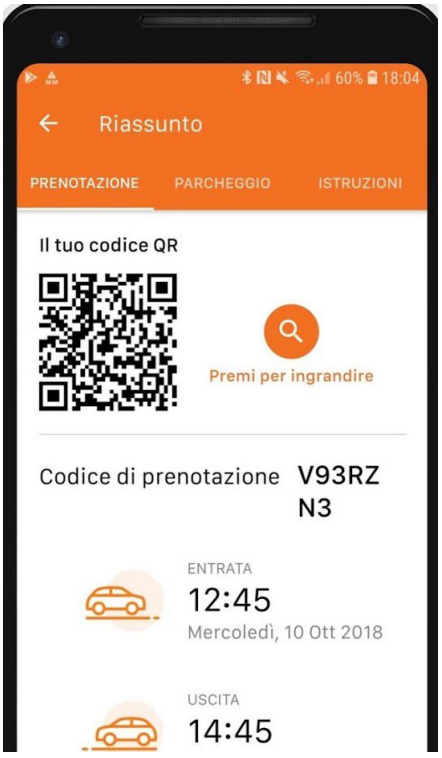

Figura 1.6: Schermata di prenotazione nell'app Parclick

1.7 Il problema della ricerca del parcheggio per i veicoli commerciali

Nei CBDs (Central business districts) ad alta densità di popolazione, lo spazio ai margini della strada dedito alla sosta è una risorsa scarsa con elevata domanda da parte di una varietà di utenti, in questi luoghi, i veicoli commerciali "competono" direttamente con veicoli passeggeri per parcheggiare negli spazi. Le politiche di gestione di queste zone influenzano la congestione stradale, l'attività delle aziende, l'estetica urbana e la sicurezza ed il comfort dei pedoni. Generalmente, la risposta a questo problema è la promozione del ricambio dei parcheggi utilizzando il controllo dei limiti di tempo e del prezzo della sosta. Tariffe più alte sono approvate da coloro che ritengono che i limiti di tempo siano difficili da monitorare e far rispettare. Alcuni sostengono che i parchimetri possono creare posti vacanti dirigendo una parte dei conducenti in parcheggi non in strada; tuttavia, i ricavi generati possono essere spesi per migliorare i quartieri tariffati.
Chiaramente, l'attuazione di una qualsiasi politica che riguardi il parcheggio dei veicoli passeggeri influisce indirettamente sulle operazioni di consegna dei veicoli merci anche se non sono il target-group. Oltre all'effetto indiretto delle politiche di parcheggio dei veicoli passeggeri per i veicoli merci, anche le normative sulle zone di carico e le restrizioni sui trasporti influiscono direttamente sulle consegne di merci. [13]

1.7.1 La piattaforma ParkUnload

La piattaforma sperimentale PARKUNLOAD, ospitata nel cloud e basata sulla tecnologia IoT (beacon con bluetooth), consente alle città di regolare e controllare in modo flessibile ed efficiente le zone di carico e altre aree di parcheggio limitate utilizzando segnali stradali intelligenti e applicazioni mobili per conducenti e vigilanti. PARKUNLOAD aumenta la rotazione dei mezzi nel parcheggio e la disponibilità di spazio (+20%), riduce il parcheggio indisciplinato (-50%) e il tempo di ricerca del parcheggio visualizzando

l'occupazione stimata in tempo reale. [9]

L'applicazione per corrieri

Quando l'autista si avvicina al segnale stradale (almeno 15 metri), l'app visualizza automaticamente il permesso di parcheggio e il tempo massimo di parcheggio per il veicolo selezionato, in base a diversi criteri: posizione della zona di carico, ora del giorno, profilo del conducente, tipo di veicolo ed emissioni. L'app visualizza il tempo di parcheggio rimanente e notifica la scadenza del tempo al conducente. Sono inoltre disponibili funzionalità avanzate per avviare / interrompere le operazioni di parcheggio utilizzando widget e comandi vocali (a mani libere).
L'app visualizza in tempo reale sia l'occupazione stimata del parcheggio che le condizioni di parcheggio del veicolo nelle zone di carico vicine (Figura 1.7), al fine di aiutare l'autista nella pianificazione della rotta. I servizi cloud della piattaforma PARKUNLOAD registrano sia la durata che lo stato di occupazione delle aree di parcheggio nelle aree di carico della città. **L'applicazione per vigilanti**
L'agente incaricato, per controllare in modo efficiente che i veicoli abbiano il permesso di parcheggio e il tempo rimanente di stazionamento nelle zone di carico della città, si deve posizionare vicino al segnale stradale verticale dell'area di parcheggio ed aprire l'app PARKUNLOAD per gli agenti. L'app visualizza il tempo di parcheggio reale di tutti i veicoli con un ticket in corso o scaduto nell'area di parcheggio controllata. L'app visualizza, inoltre, utilizzando un elenco o una mappa, i dettagli sulla scadenza del tempo di parcheggio dei veicoli nelle aree di parcheggio limitrofe.

Il backoffice per la città

Lo strumento di analisi statistica di PARKUNLOAD consente alle città di modellare le zone di parcheggio in base all'offerta e alla domanda reale (creando pattern) in aree urbane densamente popolate, in base ai big data.

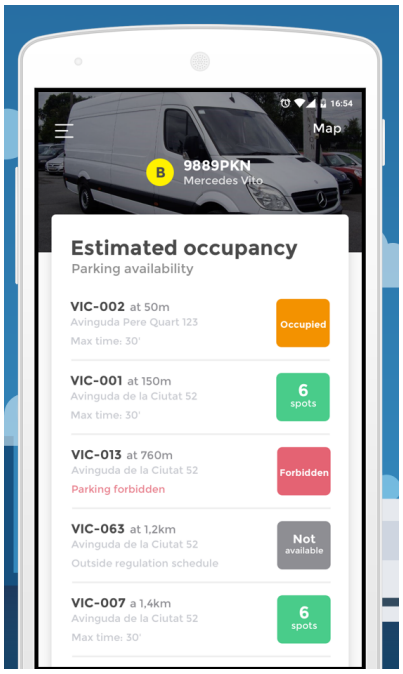

Figura 1.7: Schermata con le aree di parcheggio vicine, in Parkunload

1.8 Servizi basati sulla posizione e beacon Bluetooth

Le app per smartphone da tempo utilizzano i dati sulla posizione per tentare di migliorare l'esperienza mobile; tuttavia, ci sono sempre stati dei limiti a ciò che possono fare. A volte, determinati ambienti bloccano segnali cellulari e ciò può rendere difficile l'individuazione dei dispositivi usando il GPS; anche il GPS, però, non è in grado di tracciare la posizione esatta del dispositivo ad un livello preciso. I beacon forniscono una soluzione a questo problema usando Bluetooth Low Energy (BLE) per consentire ai sensori di rilevare - a pochi centimetri - quanto è vicino uno smartphone. Potersi affidare a Connessioni Bluetooth a bassa energia e batteria che trasmettono messaggi direttamente a un tablet o smartphone è un grande passo avanti che potrebbe aprire le porte a servizi innovativi, che potrebbero migliorare lo stile di vita delle persone. I beacon Bluetooth sono trasmettitori che

usano Bluetooth Low Energy 4.0 (BLE) per trasmettere segnali che possono essere ascoltati da dispositivi compatibili o intelligenti. Questi trasmettitori possono essere alimentati da batterie o una fonte di energia fissa come un adattatore USB. Oltre ad essere economici, sono semplici da implementare e sono supportati dalla maggior parte dei sistemi operativi per dispositivi mobili.
Quando un dispositivo intelligente è in prossimità del beacon, il beacon lo riconoscerà automaticamente e sarà in grado di interagire con quel dispositivo. Bluetooth Low Energy è una tecnologia di rete wireless personale utilizzata per la trasmissione di dati su brevi distanze progettata per un basso consumo energetico a basso costo, mantenendo un raggio di comunicazione simile a quello del suo predecessore, il Bluetooth. [7]

Capitolo 2

Scenario di sviluppo

2.1 Il caso di Barcellona

Barcellona è considerata in tutta Europa come una delle storie di successo nell'ambito dello sviluppo urbano. Come seconda città della Spagna in termini di grandezza, Barcellona è cresciuta e si sta trasformando in una città "knowledge-intensive" (ad alta intensità di conoscenza, sia professionale che tecnologico-scientifica). Oltre ad essere una meta turistica ed uno dei principali porti, è anche un ottimo esempio di cluster industriali. Nel 2009, Barcellona si è posizionata al quarto posto nella classifica delle migliori città europee per la sviluppare business.
La "Strategia" di Barcellona comprende differenti progetti, raggruppati in programmi che coprono varie aree: dalla gestione di acqua, illuminazione e rifiuti all'innovazione. All'interno di questa strategia è di grande importanza l'uso "open" della tecnologia, applicata in maniera trasversale ai servizi cittadini.
Prima, la maggior parte delle soluzioni per affrontare le sfide della città non veniva divulgata, in questo modo la città era dipendente da specifiche tecnologie, prodotti o fornitori che risolvevano problemi specifici in maniera isolata senza condividere i loro dati. Adesso, grazie ai programmi che includono l'uso trasversale della tecnologia, come il 'Telecommunications Network' o l'

'Urban Platform', queste informazioni possono essere raccolte, analizzate e comunicate, rendendo più facile condividere e gestire dati e servizi della città. L'Urban Platform prevede un modello IT architetturale della città, replicabile ed open source, formato da tre livelli [14][15]:

- Un livello più basso, che raccoglie tutti i cosiddetti 'raw data' prodotti dalla città; questi provengono però da differenti fonti: i sensori, il sistema di dati del Consiglio Comunale, il sistema informativo della città (infrastrutture, mobilità, ecc.) e dati dei Social Networks e dal Web 2.0 (gli utenti sono sia fornitori che fruitori di informazioni).

- Un cosiddetto "City Operating System", ovvero la componente intelligente dell'Urban Platform, basato su tre elementi: un City Model che guida il City Operating System; una repository globale dove viene immagazzinato tutto lo storico delle informazioni della città; ed un processo di trattamento delle informazioni che, basandosi sul City Model, applica l'intelligenza ad un insieme delle informazioni memorizzate.

- Un livello più alto formato da applicazioni e centri di controllo che hanno lo scopo di visualizzare i dati e trovare per essi degli impieghi.

L'efficace gestione della distribuzione della merce è diventata una grande sfida per la mobilità, soprattutto negli ultimi anni a causa del boom dello shopping online e del conseguente aumento dei veicoli che consegnano gli acquisti a casa. Nella città di Barcellona sono stati adibiti alla distribuzione urbana delle merci (DUM) circa novemila posti (Figura 2.1). L'obiettivo dei parcheggi regolamentati come AREE DUM è di fornire, nei giorni feriali dalle ore 8 alle 20, una zona di stazionamento, molto vicina al punto di destinazione, a tutti quei veicoli che devono distribuire merci, per un tempo limitato di 30 minuti. L'imposizione di questo lasso di tempo limite ha l'obiettivo di dare la massima rotazione a queste aree e, quindi, servire il massimo numero possibile di camion e furgoni che eseguono quest'operazione ogni giorno. La convalida telematica è obbligatoria e può essere fatta sia attraverso l'applicazione areaDUM sia inviando un SMS.

Figura 2.1: Un cartello che indica la regolamentazione di quel parcheggio come area DUM

2.2 AreaDUM, un'applicazione mobile

L'applicazione è stata lanciata nel 2014 con l'obiettivo di sostituire il classico disco orario e gestire meglio i tempi nelle aree di carico e scarico della città. Il primo passo è registrarsi come utenti, poi, una volta parcheggiati, premere il pulsante "start parking". Quindi l'app avvia un conto alla rovescia di 30 minuti e, prima che scada il tempo, emette un avviso per ricordare che i minuti stanno per scadere. Quando il veicolo viene rimosso è necessario selezionare " finish parking " nell'applicazione (Figura 2.2). Secondo il direttore della mobilità del consiglio comunale, Adrià Gomila, "questo sistema dovrebbe consentire al controllo dei 30 minuti di essere molto più efficace". Infatti, l'applicazione permetterà di conoscere l'uso reale fatto dai trasportatori di merci delle aree di carico e scarico. In questo modo, secondo Gomila, "possiamo sapere se queste aree sono sovra-occupate o sottoutilizzate", e così, aggiunge, "adattarsi alle esigenze del settore".

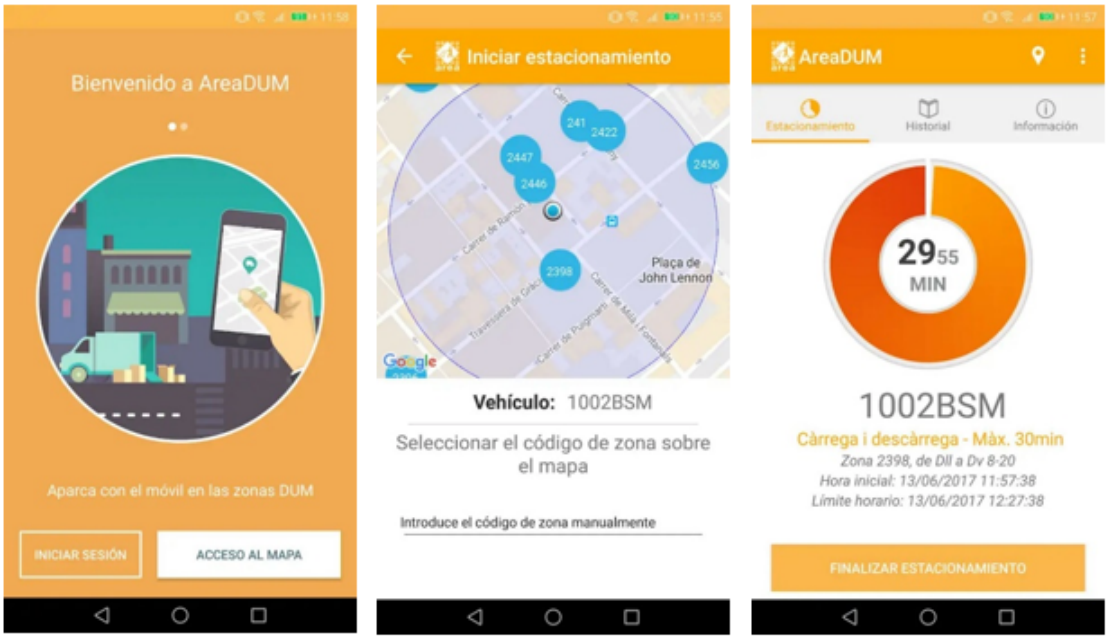

Figura 2.2: Alcune schermate dell'app AreaDUM

2.3 Travelling salesman problem e vehicle routing problem

Al problema del TSP è associabile un grafo G=(V,A), in cui V è l'insieme degli n nodi (o città) e A è l'insieme degli archi (o strade) (Figura 2.3); si indica con cij il costo dell'arco per andare dal nodo i al nodo j. Il TSP è simmetrico se cij = cji $\forall$ (i,j), altrimenti si dice asimmetrico: nel TSP simmetrico, la distanza tra due città è la stessa in ogni direzione opposta, formando un grafo non orientato. Questa simmetria dimezza il numero di possibili soluzioni. Nel TSP asimmetrico, i percorsi potrebbero non esistere in entrambe le direzioni o le distanze potrebbero essere diverse, formando un grafico diretto. Le collisioni del traffico, le strade a senso unico e le tariffe aeree per le città con tariffe di partenza e di arrivo diverse sono esempi di come questa simmetria si potrebbe abbattere. Non esistono algoritmi efficienti per la risoluzione del TSP, l'unico metodo di risoluzione è rappresentato dall'enumerazione totale, ovvero nell'elaborazione di tutti i possibili cammini

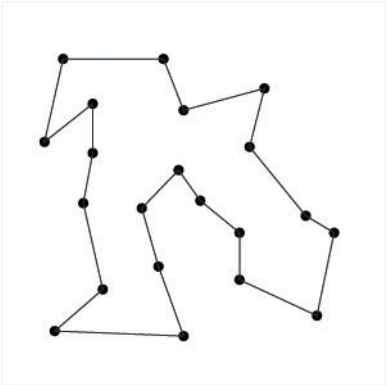

Figura 2.3: Grafo del TSP

sul grafo per la successiva scelta di quello migliore. Tuttavia, la complessità dell'operazione la rende impraticabile per grafi di dimensioni comuni nei problemi reali: in un grafo di n nodi, bisognerà calcolare, nel caso peggiore in cui ogni nodo è connesso con tutti gli altri, n! possibili cammini. Il TSP diventa esponenzialmente più complesso da risolvere all'aumentare dei punti che il corriere deve visitare.
Il TSP rientra nella classe dei problemi NP-Completi. Il metodo che spesso viene utilizzato per risolvere questo tipo di problemi è la progettazione di algoritmi euristici, cioè algoritmi che producono soluzioni probabilmente buone, ma impossibili da provare essere ottimali; che hanno quindi un'"alta" probabilità di produrre una "buona" soluzione "velocemente".
Nella realtà odierna però, solitamente, la compagnia possiede più di una singola risorsa per effettuare le consegne, si ha quindi un intero team di corrieri a cui assegnare rotte differenti che, insieme, coprano tutti i punti; questo secondo problema viene chiamato Vehicle Routing Problem (Figura 2.4). Il Vehicle Routing Problem (VRP) è un tipico problema operativo nelle reti di distribuzione, e consiste nello stabilire i percorsi di una serie di veicoli per servire un insieme di clienti. Dato un insieme di veicoli con determinate caratteristiche che devono visitare un insieme di clienti (anche essi con determinate caratteristiche) distribuiti all'interno di una rete di trasporto a partire da uno (o più) depositi centrali. Possibili applicazioni del VRP sono

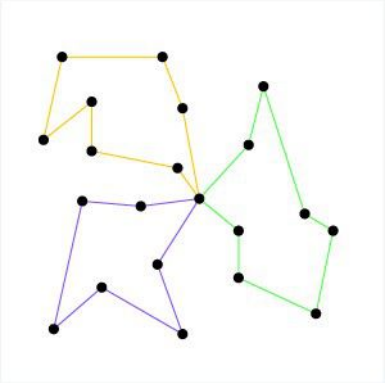

Figura 2.4: Grafo del VRP

frequenti in problemi logistici e distributivi di dettaglio. Per esempio, si può pensare a contesti applicativi in cui si debba distribuire al dettaglio (oppure provvedere alla raccolta) un certo bene.

2.3.1 Le variabili da considerare nell'applicare TSP e VRP nel mondo reale

Immaginando di dover organizzare un operazione di consegna, si possono notare alcune altre variabili che si rivelano cruciali per il servizio che si offre: punti di inizio e fine (il numero di depositi da cui iniziare le spedizioni), il numero di viaggi per ogni veicolo (se i conducenti devono tornare nei magazzini a rifornirsi di merce per continuare le loro consegne), la priorità della consegna (un particolare cliente ha la precedenza e quindi si potrebbe voler consegnare il suo prodotto per primo), la capacità di ogni veicolo; questi vincoli possono influenzare la computazione della rotta.
Gli studi dimostrano che la cosiddetta "last mile delivery" rappresenta il 28% del costo dello spostamento di un articolo dalla fabbrica al suo acquirente. Utilizzando un route optimizer le aziende possono ridurre la lunghezza dei loro percorsi fino al 40%. Ottenendo un risparmio sui costi di carburante e manutenzione e l'opportunità di raggiungere ancora più clienti in una volta.

2.4 L'obiettivo del progetto

Alla luce delle precedenti premesse, l'intento è quello di creare un'applicazione mobile per corrieri, che si basi sul funzionamento del route optimizer CIGO!fleet, con features però mirate all'individuazione dei parcheggi disponibili (in questo caso le aree DUM) nella zona in cui si trova il punto di consegna. Il corriere, con il suo smartphone, potrà quindi, dopo aver scaricato le rotte più attuali dal server della sua compagnia, visualizzare i parcheggi più vicini ad ogni singola fermata, trovati in tempo reale direttamente dal suo dispositivo attraverso l'applicazione mobile.
L'idea di questa integrazione ad un applicazione già avviata è nata notando che era stata progettata pensando ad uno scenario di applicazione utopistico in cui per ogni punto di consegna era disponibile un parcheggio nelle strette vicinanze; ma può succedere che il corriere arrivi con il mezzo davanti all'indirizzo di destinazione, guidato dal navigatore satellitare, per poi dover cercare nelle zone limitrofe un parcheggio, perdendo inevitabilmente tempo. Si deve inoltre tenere conto delle peculiarità della città di Barcellona dove, soprattutto nel suo centro storico che fa del turismo uno dei suoi pilastri economici, è presente un'innumerevole quantità di piccole e medie attività commerciali. Queste attività vanno dal commercio al dettaglio, all'attività alberghiera e della ristorazione, settori che necessitano di un assiduo contatto con i loro fornitori, questo implica un grande numero di camion nella "Città Vecchia" nelle medesime fasce orarie; ciò, insieme alle ridotte dimensioni delle antiche strade ed alle rigide regole sulla mobilità non facilita, per i corrieri, la circolazione e la ricerca di un punto di stazionamento momentaneo.

Capitolo 3

Sviluppo del progetto

3.1 Progettazione e ambiente di sviluppo

Per questo progetto, è stato inizialmente strutturato e perfezionato un algoritmo di ricalcolo delle rotte in base alla dislocazione delle aree DUM e, successivamente, si è proceduto a sviluppare un'applicazione mobile, per dispositivi Android, che utilizza l'algoritmo creato; per questa ragione sono state prodotte due differenti versioni, provvedendo a "snellire" la parte computazionale che sarebbe entrata a far parte dell'app. Si sono compiute due scelte per rendere il processo di sviluppo più veloce:

1 Nonostante fosse previsto che le rotte venissero inserite in un database sul server della compagnia e che il corriere, con il suo smartphone attraverso l'applicazione vi possa accedere, per facilitare i test, la rotta è stata inserita in un file csv con indirizzo, coordinate ed altre informazioni per ogni punto del percorso.

2 Per il calcolo della distanza tra due punti si è adottato il metodo della distanza euclidea perché, dopo aver sperimentato le API di google maps o di open street map, si è notato che le richieste http rallentavano di molto il servizio, seppur più affidabili perché il valore ottenuto teneva conto della presenza di edifici, di limitazioni alla circolazione e della morfologia della città; verranno comunque riprese nell'applicazione.

Per quanto riguarda l'algoritmo, il linguaggio scelto è Java e l'ambiente di sviluppo che è stato utilizzato per la progettazione ed il testing è Netbeans IDE 8.2 , inoltre la libreria Sparksee 5.2 fornisce il GDBMS di supporto.

3.2 Open Data Bcn

Il portale "Open Data Bcn" del comune di Barcellona è nato con l'intento di sfruttare al massimo le risorse pubbliche disponibili, esponendo le informazioni generate o custodite da organismi pubblici, consentendo il loro accesso e riutilizzo per il bene comune ed a beneficio delle persone e delle entità interessate. E' possibile trovare dati su molti argomenti, provenienti da fonti di vario genere; ma per questo caso si ha la necessità dell'elenco delle aree DUM dislocate nella città di Barcellona (file 2018_05_TRAMS.CSV), presente nella sezione "catalogo dei dataset" (http://opendata-ajuntament.barcelona.cat/data/es/dataset/trams-aparcament-superficie).
Il dataset contiene le informazioni (progressivo, latitudine, longitudine, indirizzo, numero di posti) non solo delle aree DUM (gialle) ma anche delle altre aree di parcheggio regolate di Barcellona (verdi e blu), si estraggono quindi solo le gialle, in maniera tale da ottenere un file .csv contenente esclusivamente le informazioni delle aree di nostro interesse (Figura 3.1). Si ottiene così un elenco di oltre 2000 aree DUM, un numero di dati significativo, soprattutto se si considera la capacità computazionale limitata di uno smartphone. Inoltre, si arriva alla conclusione che, il database (che conterrà inizialmente sia aree DUM che punti della rotta e, in una seconda versione, solo aree DUM) risulterà essere una base di dati in cui, per questo scopo, si svelano più rilevanti le relazioni tra i dati piuttosto che i dati stessi; l'informazione d'interesse è quindi contenuta nell'esito tra la loro comparazione. Per questo motivo si è scelto di proseguire l'implementazione del progetto con un database a grafo.

51	41.378372770695	2.170089910224	1059.- RAVAL, 41, RBLA	5
52	41.379448789986	2.168072027874	1060.- RAVAL, 4 bis, RBLA	4
53	41.376862371409	2.16719509379	1061.- REINA AMALIA, 33, C	5
54	41.380545639166	2.166710064878	1062.- RIERA ALTA, 12, C	4
55	41.380403993501	2.165183100651	1063.- RIERA ALTA, 43, C	3
56	41.381767836786	2.163609146982	1065.- SANT ANTONI, 58, RDA	4
57	41.382714780469	2.163735869992	1066.- SANT ANTONI, 66, RDA	4
58	41.377261129083	2.17137585763	1067.- SANT OLEGUER, 18, C	4
59	41.379475582669	2.170101039884	1068.- SANT RAFAEL, 7, C	4
60	41.379388662694	2.169898744539	1069.- SANT RAFAEL, 15, C	1
61	41.379250725437	2.169592869463	1070.- SANT RAFAEL, 17, C	3
62	41.385000334211	2.165864186672	1071.- TALLERS, Front 72, C	9

Figura 3.1: Porzione del file csv contenente i dettagli sulle aree DUM

3.3 Graph Database

Un modello di databse a grafo è un modello in cui le strutture dati per lo schema e/o le istanze sono modellate come grafi (o generalizzazioni di essi), dove la manipolazione dei dati è espressa da operazioni graph-oriented (es. un linguaggio di query per grafo) e possono essere definiti sulla struttura del grafo appropriati vincoli di integrità. Una delle principali caratteristiche della struttura a grafo è la semplicità di modellare dati non strutturati; inoltre, la separazione tra schema e dati (istanze) è meno marcata rispetto al classico modello relazionale. I nodi rappresentano le entità in un certo dominio e gli archi (linee che collegano i nodi ad altri nodi) rappresentano le relazioni tra queste entità (Figura 3.2). I nodi sono l'equivalente del record, della relazione o della riga in un database relazionale o del documento in un document database. Alcuni database a grafo richiedono la definizione di uno schema per il proprio grafo, ad es. definizione di etichette o nomi per i nodi, gli archi e le proprietà prima di inserire dati, mentre altri database consentono di operare senza uno schema fisso. Nell'esempio raffigurato nella Figura 3.3:

- I nodi appartengono a tre domini differenti: clienti (blu), ristoranti (viola), categorie (verde).
- Gli archi che connettono i nodi hanno tre label che li descrivono: visita A, è di genere B, è amico di C.

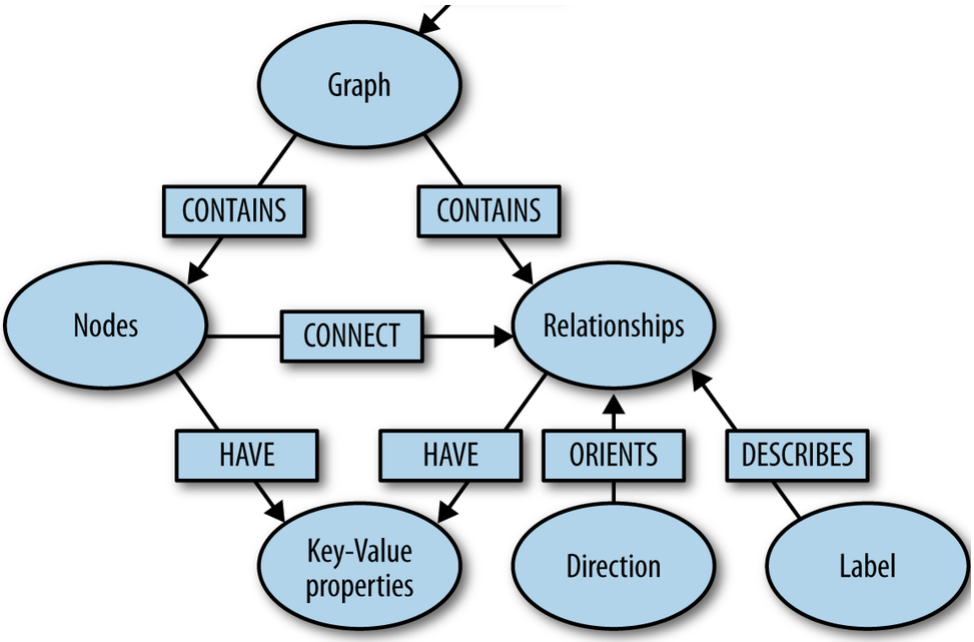

Figura 3.2: Come funziona un graph database

- Sia nodi che archi hanno delle proprietà individuate come coppie chiave-valore: nome e giudizio.

Non si presentano quindi differenze nelle informazioni che si possono acquisire nel modello di dati a grafo che non si potrebbero acquisire in un modello di dati relazionale tradizionale. Semplicemente descrivendo le relazioni tra le tabelle usando chiavi esterne, oppure si possono descrivere le proprietà di una relazione con una tabella di join. La differenza fondamentale tra questi modelli di dati è il modo in cui i dati sono organizzati e accessibili. Il riconoscimento degli archi come "dati rilevanti" accanto ai nodi nel modello di dati a grafo consente al database engine, un componente software che un sistema di gestione del database (GDBMS) utilizza per creare, leggere, aggiornare e cancellare (CRUD) i dati da un database, sottostante di scorrere molto rapidamente in qualsiasi direzione attraverso reti di nodi e archi per soddisfare le query dell'applicazione, un processo noto come attraversamento.[1] [2]

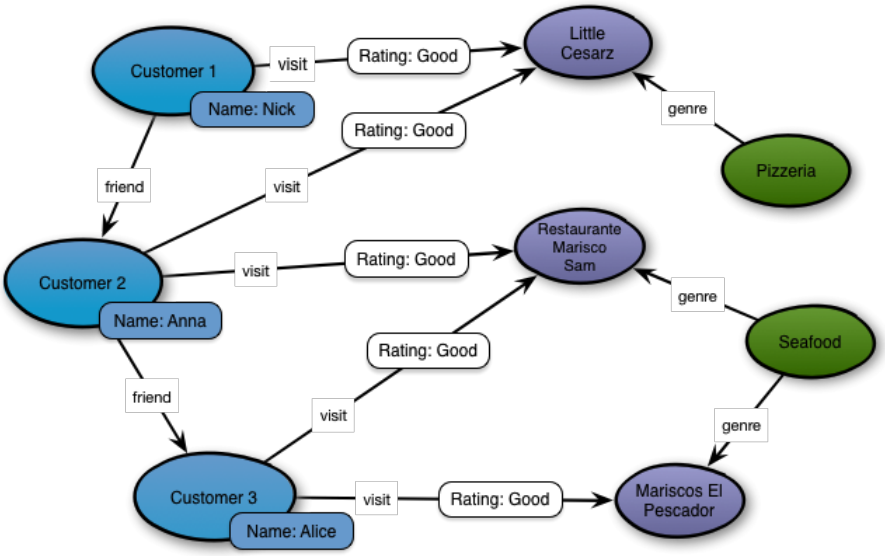

Figura 3.3: Un esempio di utilizzo di un graph database

3.4 Sparksee

Sparksee (conosciuto come DEX), prodotto nato dalla ricerca effettuata presso DAMA-UPC (gruppo Data Management presso l'Università Politecnica della Catalogna), è un DBMS a grafo ad alte prestazioni e scalabile. Il suo sviluppo ha avuto inizio nel 2006 e la sua prima versione fu resa disponibile nel 2008; esiste una community version gratuita, per scopi accademici o di valutazione, disponibile per il download, limitata a 1 milione di nodi, senza limiti sugli archi. Nel marzo 2010 è stata creata una spin-off, Sparsity-Technologies, presso l'UPC per commercializzare e fornire servizi alle tecnologie sviluppate da DAMA-UPC. DEX cambiò nome in Sparksee alla sua 5a release nel 2014.

3.4.1 Caratteristiche

Un grafico DEX è un "Labeled Directed Attributed Multigraph":

- Labeled perché i nodi e gli archi in un grafico appartengono a determinati tipi.

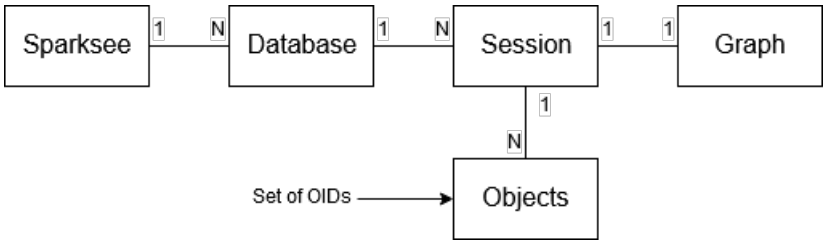

Figura 3.4: Sparksee API class diagram. Sparksee può gestire più database, i cui grafi possono essere letti uno alla volta, purché all'interno di una sessione. [3]

- Directed perché supporta archi orientati e non orientati.
- Attributed perché sia i nodi che gli archi possono avere attributi.
- Multigraph che significa che possono esserci più archi tra gli stessi nodi, anche se provengono dallo stesso tipo di arco.

Una delle sue caratteristiche principali solo le sue performance di archiviazione e di reperimento delle informazioni per grafi di grandi dimensioni (nell'ordine di miliardi di nodi, archi e attributi). Si basa sul partizionamento verticale e bitmaps (per ridurre lo spazio occupato). Con la rappresentazione bitmap ogni nodo o arco è identificato da un unico ed immutabile oid, ed è memorizzato in una struttura bitmap. Ogni posizione in una bitmap corrisponde all'iod di un oggetto. Principali dettagli tecnici di Sparksee:

- Linguaggio di programmazione: C++ API: Java, .NET, C++, Python, Objective-C.
- Compatibilità con i SO: Windows, Linux, Mac OS, iOS.
- Persistenza: Disk.
- Transazioni: full ACID.
- Recovery Manager.

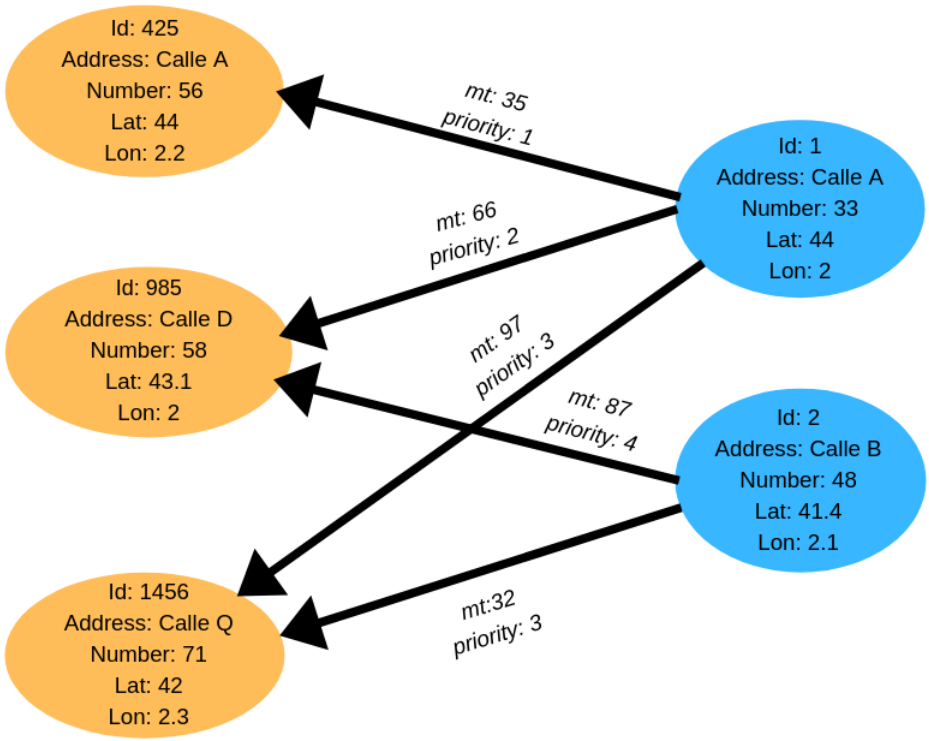

Figura 3.5: Un esempio della struttura del grafo con la prima implementazione dell'algoritmo

3.5 Prima implementazione dell'algoritmo

In questa soluzione, inizialmente vengono dati in input e memorizzati nel database a grafo Sparksee i punti della rotta prima e le aree DUM dopo; per ogni area (nodo Ai), nel momento in cui questa viene letta in input, viene calcolata la sua distanza (euclidea) da ogni punto del percorso dato (nodo Rj) e questo valore diventa un attributo dell'arco che sarà creato tra i nodi Ai ed Rj (Figura 3.5). Successivamente, si procede con la ricerca delle aree di parcheggio ottimali rispetto la rotta indicata. Per ogni punto del percorso si visitano tutti gli archi uscenti (che lo collegano con le aree DUM)

e viene assegnata una priorità (dal più vicino al più lontano), sottoforma di attributo, ad ogni arco che collegano il punto ad un area. Infine, viene restituita l'area più vicina, ovvero quella il cui arco entrante dal nodo Ai ha l'attributo 'priorità' settato a 1.

3.5.1 Struttura della classe

I principali metodi sono, in ordine di invocazione:

1 nodeTypeCreator: crea i nodi di tipo DUM e POINT, rappresentanti rispettivamente i parcheggi ed i punti di consegna, ed assegna loro degli attributi; a seguire un esempio di codice:

```
dumType = g.newNodeType("DUM");
dumIdType = g.newAttribute(dumType, "ID", DataType.Long,
    AttributeKind.Unique);
dumLatType = g.newAttribute(dumType, "LAT", DataType.Double,
    AttributeKind.Basic);
```

2 edgeTypeCreator: crea gli archi di tipo DISTANCE, che rappresentano la distanza tra un area DUM ed un punto della rotta, ed assegna loro gli attributi; a seguire un esempio di codice:

```
distanceType = g.newRestrictedEdgeType("DISTANCE", pointType,
    dumType, false);
mtDistType = g.newAttribute(distanceType, "MT",
    DataType.Integer, AttributeKind.Basic);
```

3 pointOfTheRouteLoader: legge i punti di una rotta da un file .csv e li carica sotto forma di nodi nel database a grafo creando due array che rappresentano il tipo di attributo (attrs) e la posizione della sua colonna nel file (attrPos), e passandoli come parametri all oggetto NodeTypeLoader.

4 areasDumLoader: legge le aree DUM da un file .csv e le carica nel database, allo stesso tempo memorizza le distanze con i punti della rotta, creano gli archi con saveDistance() che viene invocato per ogni area DUM;

5 saveDistance: crea l'arco tra un punto di consegna (iterando) e l'area DUM in input, solo se il valore calcolato per la distanza (euclidea) è minore della massima distanza prefissata;

6 distGeo: calcola la distanza euclidea tra due punti, date le loro coordinate;

7 apiDist (in alternativa a distGeo): calcola la distanza tra due punti date le loro coordinate, considerando la reale superficie della terra (strade, edifici, ecc), con una richiesta http al servizio Google Maps;

8 orderAreaDum: per ogni nodo di tipo POINT estrae gli archi che lo connettono ai nodi di tipo DUM e, successivamente, invoca orderEdges per ordinarli;

9 orderEdges: applica l'algoritmo di bubble sorting agli archi ricevuti in input, assegnando la priorità corretta in base al valore dell'attributo distanza;

10 nearestDum: stampa i punti della rotta con le aree DUM più vicine a ciascuno.

3.5.2 Valutazioni

Questa prima soluzione, che può sembrare la più naturale, non è però efficiente dal punto di vista computazionale. Infatti, per ogni punto della rotta si procede con il calcolo della distanza con tutte le aree DUM, memorizzando (tramite un arco) però solo quelle la cui distanza non supera un certo limite prefissato (es. 1 km), si deve quindi trovare un modo per evitare richieste http che si rivelerebbero poi inutili, restringendo il raggio in partenza.

Inoltre, l'intero processo verrebbe svolto direttamente nell'applicazione mobile, che conterrebbe come dati di partenza, già al suo interno, un elenco di punti di consegna e l'elenco delle aree DUM ordinato casualmente; a partire da questi dati deve essere poi considerato il tempo di creazione dei nodi e del loro inserimento nel database a grafo. Si deve tenere presente che le rotte vengono fornite al corriere di volta in volta, ciò porterebbe, secondo questa implementazione (prima inserimento punti, poi inserimento aree DUM), a creare ogni volta un nuovo grafo nel database e ad eventualmente eliminare quelli precedenti (se inutili); di conseguenza si continuano a creare ed inserire nodi per le aree DUM quando si potrebbe avere per queste un database permanente.
Con questa seconda alternativa si avrebbe un unico database a grafo, inizialmente composto solo dai nodi rappresentanti i parcheggi, a cui verrebbero aggiunti quelli che identificano i punti della nuova rotta (ogni qualvolta se ne presenti una) ed eliminati quelli che identificano invece i punti della rotta precedente, insieme agli archi creati che indicano la distanza punto - area Dum.
In entrambi i casi, si nota che la memorizzazione dei punti della rotta nel database risulta superflua, in quanto gli archi nel grafo (tra punti ed aree) verranno sfruttati solo la prima volta per scoprire quali sono le aree DUM più vicine a ciascun punto, ed il loro ordine. Dopodiché si può procedere alla memorizzazione della rotta "ottimizzata", con i dettagli dei punti di consegna e dei rispettivi parcheggi, in un database distinto dal precedente, evitando quindi di dover andare a "pulire" il graph database dagli oggetti ormai non più utili.

3.6 Seconda implementazione dell'algoritmo

Nell'ottica di dover implementare il processo di computazione in un'applicazione mobile, è stata pensata una seconda alternativa, che permette di snellire notevolmente la parte di processo intra-app. Mentre prima si aveva

un'unica classe che creava il database e calcolava i parcheggi più vicini ad una data rotta, con questa implementazione è possibile separare i due processi, riducendo notevolmente i tempi.

3.6.1 RB Tree

Prima di trattare nello specifico la struttura dati ad albero RB è opportuno introdurre gli alberi di ricerca binari. Un albero binario è un albero i cui nodi hanno grado compreso tra 0 e 2. Per albero si intende un grafo non diretto, connesso e aciclico mentre per grado di un nodo si intende il numero di sottoalberi del nodo, che è uguale al numero di figli del nodo.
La definizione formale di un albero binario di ricerca è quella di un albero binario T avente le seguenti tre proprietà, in cui si indica con x.key la chiave (il valore) di un nodo x $\in$ T:

1 $\forall$ x $\in$ T , x.key $\in$ C dove C è un insieme parzialmente ordinato

2 $\forall$ x $\in$ T , y $\in$ T : y $\in$ left (x) $\rightarrow$ y.key $\leqslant$ x.key dove left (x) rappresenta il sottoalbero sinistro di un nodo x $\in$ T

3 $\forall$ x $\in$ T , y $\in$ T : y $\in$ right (x) $\rightarrow$ y.key $>$ x.key dove right (x) rappresenta il sottoalbero destro di un nodo x $\in$ T.

Un binary search tree ha quindi, in particolare, le seguenti proprietà:

1 Il sottoalbero sinistro di un nodo x contiene soltanto i nodi con chiavi minori della chiave del nodo x

2 Il sottoalbero destro di un nodo x contiene soltanto i nodi con chiavi maggiori della chiave del nodo x

3 Il sottoalbero destro e il sottoalbero sinistro devono essere entrambi due alberi binari di ricerca.

Ai fini del nostro algoritmo, le operazioni di cui si ha necessità sono l'inserimento e la ricerca:

- Le proprietà 1 e 2 consentono di realizzare un algoritmo di ricerca dicotomica o binaria che, essendo confinata ai nodi posizionati lungo un singolo percorso (path) dalla radice ad una foglia, avrà un costo computazionale O(h) dove h è l'altezza dell'albero (il massimo livello delle sue foglie, indicato con il numero di archi attraversati dalla radice al nodo considerato).

- Per l'inserimento l'algoritmo è simile a quello della ricerca. In pratica si svolge una ricerca fin quando non si esce dall'albero e l'ultimo nodo attraversato prima di uscire sarà il padre del nuovo elemento inserito. A questo punto si confronta il valore del padre con quello da inserire e si inserisce adeguatamente a sinistra o destra del padre il nuovo valore. L'algoritmo ha quindi la stessa complessità algoritmica di quello di ricerca, e quindi O(h) con h la profondità dell'albero.

Il fattore di bilanciamento β(v) di un nodo v è la massima differenza di altezza tra i sottoalberi di v, un albero "perfetto" dovrebbe avere β(v) = 0 $\forall$ nodo. Quindi, un albero è perfettamente bilanciato se ha tutte le foglie al medesimo livello, ovvero se ogni foglia dell'albero ha la medesima distanza dalla radice. La condizione per tenere l'albero bilanciato è la seguente: per ogni nodo dell'albero, la differenza di altezza dei suoi sottoalberi figli deve essere al massimo di 1 ($|\beta$ (v)$| \leqslant 1$). Grazie a questa restrizione, l'altezza massima dell'albero, ossia la più grande distanza tra la radice e le foglie, è logaritmica nel numero dei nodi. Ciò permetterebbe ad una struttura dati di compiere l'inserimento, la ricerca e l'eliminazione di un elemento in O(log n).
Un albero rosso-nero (Figura 3.6) è un albero di ricerca binario bilanciato in cui:

- ogni nodo ha un colore (rosso o nero) ad esso associato (oltre alla sua chiave ed ai figli sinistro e destro)

- valgono i seguenti 3 vincoli:

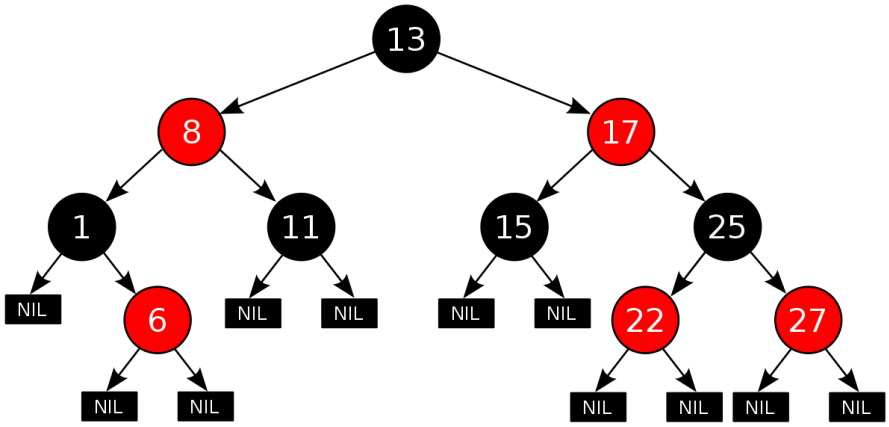

Figura 3.6: Esempio di albero RB

1 (proprietà root) La radice dell'albero RB è nera

2 (proprietà red) I figli di un nodo rosso sono neri

3 (proprietà black) Ogni percorso dalla radice alla foglia di un albero contiene lo stesso numero (la "altezza nera") dei nodi neri

4 le foglie sono nere.

Questi vincoli rafforzano una proprietà critica degli alberi rosso-neri: che il cammino più lungo dal nodo root a una foglia è al massimo lungo il doppio del cammino più breve. Ne risulta dunque un albero fortemente bilanciato. Poiché le operazioni di ricerca di un valore, inserimento e cancellazione richiedono un tempo di esecuzione nel caso peggiore proporzionale all'altezza dell'albero, questo limite superiore teorico sull'altezza rende gli alberi rosso-neri molto efficienti nel caso peggiore, al contrario di quanto accade con gli ordinari alberi binari di ricerca. Le operazioni di lettura su un albero rosso-nero non richiedono modifiche rispetto a quelle utilizzate per gli alberi binari di ricerca, poiché gli alberi rosso-neri sono una loro specializzazione. Al contrario, durante la modifica di un albero RB è possibile che le condizioni di bilanciamento risultino violate, si può agire modificando i colori nella zona di violazione ed operando dei ribilanciamenti dell'albero tramite rotazioni a sinistra e a destra. Nonostante l'inserimento e la cancellazione siano opera-

zioni complicate, i loro tempi di esecuzione rimangono dell'ordine di O(log n) perché risalendo si ripercorre l'albero al più per la sua altezza.[4] [5]
Per l'inserimento:

1 Si ricerca la posizione usando la procedura degli ABR

2 Si colora il nuovo nodo di rosso, per mantenere il vincolo dell'altezza nera

3 Si procede verso l'alto per ripristinare il vincolo 2, spostando le violazioni verso l'alto rispettando il vincolo 3; si hanno vari casi per l'inserimento 4. Al termine, si colora la radice di nero per soddisfare la proprietà della root nera; poiché questo inserimento aggiunge un nodo nero a ciascun cammino, la proprietà 3 (tutti i cammini a partire da un dato nodo verso le sue foglie contengono lo stesso numero di nodi neri) non è violata.

3.6.2 Prima fase

In questa prima parte, le coordinate le aree DUM sono sempre memorizzate all'interno del database a grafo come nodi, ma vengono però duplicate in due grafi differenti, entrambi organizzati come alberi RB (alberi binari di ricerca bilanciati); nel primo i nodi vengono ordinati in base al valore della latitudine (Figura 3.7), nel secondo in base a quello della longitudine. Utilizzando sistematicamente gli archi, viene replicata nel database la struttura dati ad albero per connettere i nodi (questo permette di navigare efficacemente attraverso la stessa). Con questa prima parte è quindi necessaria una sola esecuzione (in un calcolatore) per creare i due database, che verranno poi caricati nella memoria interna all'applicazione mobile (direttamente da Android Studio) in maniera tale che chiunque dovesse scaricare l'applicazione mobile possegga già i due database con le aree DUM. Questi ultimi verranno poi utilizzati ogni volta che un corriere scarica una nuova rotta e decide di trovare i relativi parcheggi "ottimali".

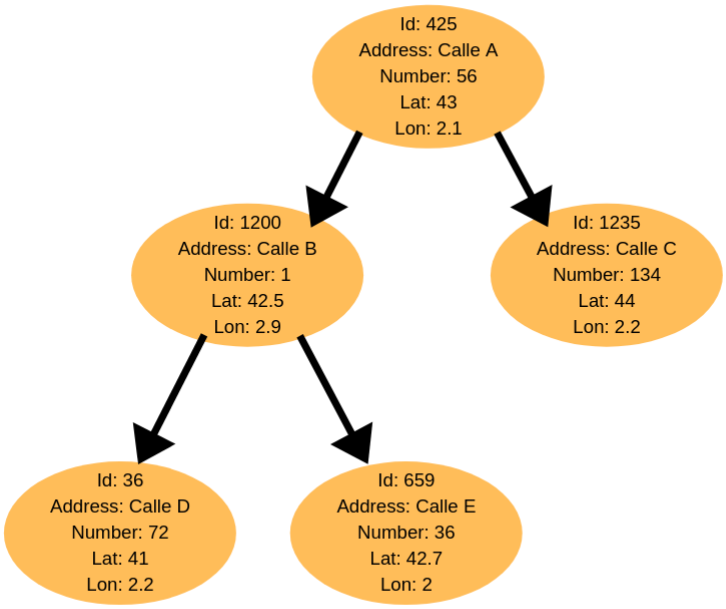

Figura 3.7: Esempio di grafo contenente le aree DUM ordinate per latitudine

Nella figura è mostrato un possibile esempio di organizzazione di un frammento del graph database contenente le aree DUM ordinate a seconda del valore della latitudine in un albero Red-Black.

Struttura delle classi

Le classi che costituiscono questa prima parte sono due:

- RBTree.java: che implementa al suo interno la struttura e le operazioni di un albero RB
- TreeCreator.java: che crea il graph database delle aree DUM collegando i nodi, attraverso gli archi, come se fossero in un albero RB.

Si procede eseguendo quindi TreeCreator.java, la quale si deve trovare nello stesso pacchetto di RBTree.java per avere il supporto di quel tipo di oggetto.

RBTree.java

Per quanto riguarda la classe RBTree.java, il costruttore è dato da RBTree()root = null;, la sua unica variabile di istanza è quindi la radice dell'albero, che è un elemento di tipo Node: la classe Node, rappresenta un'area DUM, e ogni oggetto di tipo Node ha come variabili di istanza:

- key: il valore della coordinata da tenere in considerazione per quel grafo
- id: identificativo del nodo nel database
- progressivo: identificativo dell'area DUM nel file csv
- left, right e parent: oggetti di tipo Node, quindi il riferimento ai figli sinistro e destro ed al padre
- color: il colore del nodo, 0 per nero e 1 per rosso.

Per la fase di inserimento, su ogni area DUM viene invocato da TreeCreator.java il metodo insert() che, dopo aver creato il nodo, invoca insertRec() per inserirlo nell'albero e poi spostarlo nel punto corretto con insert_case1(),

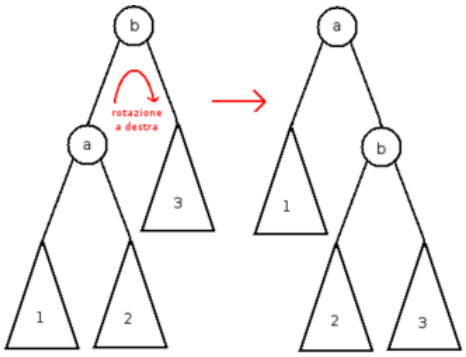

Figura 3.8: Funzionamento della rotazione

insert_case2(), insert_case3(), insert_case4(), insert_case5(), si veda https://www.geeksforgeeks.org/red-black-tree-set-2-insert. Il metodo insertRec() viene eseguito in maniera ricorsiva per inserire un nuovo nodo in base alla sua key nell'albero; insert_case1() è invece il primo caso che verifica se il nuovo nodo n è nella root dell'albero, questo viene colorato di nero per soddisfare la proprietà 2 (la root è nera). I successivi casi di inserimento verificano la posizione del nodo provvedendo a ricolorazioni e/o rotazioni (vedi Figura 3.8) e, se necessario, invocano il caso successivo. Infine, la classe RBTree.java fornisce la possibilità di ottenere i nodi dell'albero ordinati in una lista, in base alla chiave, con il metodo inorder() che inizia un attraversamento del grafo, invocando inorderRec()).

TreeCreator.java

Nella classe TreeCreator.java, all'interno del blocco main vengono creati i due graph database (per latitudine e longitudine). Prima di tutto viene creato il database per quel tipo, poi aperta una sessione su quel database e si estrae il grafo, vengono poi creati i nodi di tipo "DUM" e gli archi di tipo "parent", "left" e "right", si procede quindi al caricamento nel grafo delle aree DUM presenti nel file csv e contemporaneamente al loro inserimento in una struttura al albero RB "parallela" che rispetta quindi i vincoli degli alberi bilanciati ed infine all'aggiunta degli archi nel grafo (seguendo la struttura dell'albero RB parallelo) in maniera tale che colleghino i nodi tra loro come

se fossero organizzati secondo la struttura dati sopranominata.
Per la latitudine il procedimento è il seguente, identico a quello per la longitudine: si crea il graph database e su questo si apre una sessione, a questo punto si richiede l'oggetto grafo in cui vengono creati il tipo di nodo (area DUM) ed il tipo di arco (relazione) con i relativi attributi; infine vengono caricate le aree DUM nel databse e si riproduce la struttura al albero RB con l'utilizzo delle connessioni. In particolare, areasDumLoader() itera tra i nodi del grafo per inserirli correttamente nell'albero RB e successivamente, il metodo graphInTree() trasforma le connessioni tra i nodi dell'albero in archi del grafo; iterando tra i nodi e creando nuovi archi in base alle variabili d'istanza parent, left e right.

3.6.3 Seconda fase

La seconda parte dell'algoritmo è stata pensata per essere totalmente eseguita nell'applicazione mobile del corriere nel momento in cui questo decide di riorganizzare il suo percorso di consegne secondo la dislocazione delle aree DUM. Come mostrato nella Figura 3.9, la rotta è identificata dall'unione dei segmenti neri che rappresentano il tragitto tra un punto di consegna ed il successivo, le aree DUM vengono rappresentate con rettangoli arancioni. L'idea è quella di voler evitare di dover calcolare la distanza tra un punto della rotta e tutte le aree DUM, navigando top-down l'albero RB contenente le aree DUM ordinate per singola coordinata (latitudine o longitudine considerate separatamente calcolando la differenza, ad esempio, tra la latitudine del punto A e la latitudine dell'area 3), fino a quando non si trova l'area DUM che abbia la minore differenza per quella data coordinata; a quel punto, dato il nodo pivot, ci si sposta a destra e/o sinistra nella lista ordinata delle aree DUM per trovare altre aree con una coordinata la cui differenza con la coordinata del punto si mantenga nel range prestabilito. Dopo aver eseguito questo processo per entrambe le coordinate, si procede ad un matching, per ogni punto, tra le aree DUM risultate ottimali dal punto di vista della latitudine e quelle ottimali dal punto di vista della longitudine. Si calcolerà

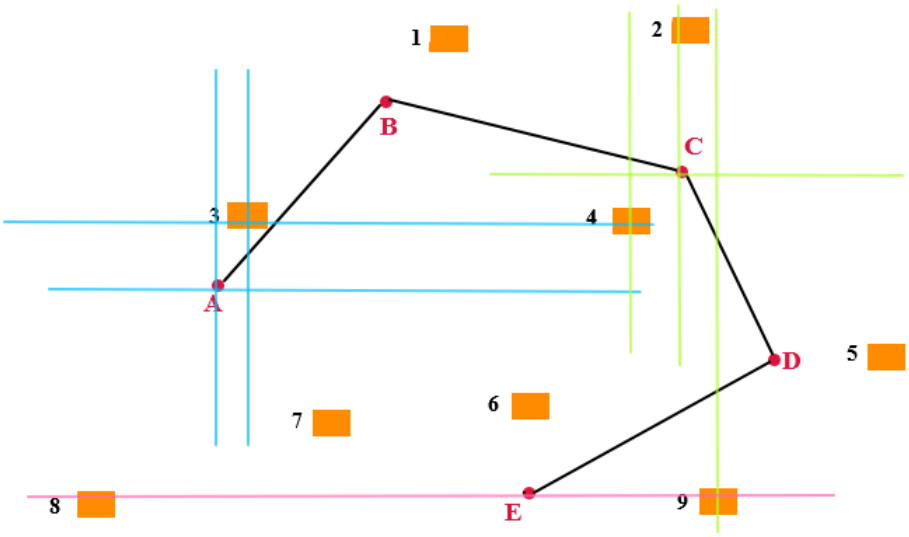

Figura 3.9: Procedimento della seconda fase

quindi la distanza tra il punto di consegna e le aree di parcheggio risultanti dall'intersezione.
Dopo aver ottenuto l'elenco delle aree relative ad ogni punto del percorso, si procede ad aggregare le aree DUM in modo da produrre un "percorso reale" composto dalle zone in cui il corriere dovrà parcheggiare; i criteri seguiti sono: le aree che sono in comune tra due punti successivi e la distanza che il corriere è disposto a coprire a piedi in caso di adozione di una soluzione aggregata.

Struttura della classe DistanceCalculator.java

Questa classe è stata creata con l'obiettivo di essere poi esportata nell'applicazione mobile. Il procedimento può essere diviso in 4 fasi: lettura della rotta da file, calcolo dei punti più vicini in termini di singole coordinate con i database delle aree ordinate per latitudine e longitudine, intersezione tra i due insiemi di aree DUM trovate per le coordinate di ogni punto della rotta e calcolo della distanza tra ciascun punto e le aree risultanti dall'intersezione

ed, infine, l'aggregazione che consiste in due casi: il primo in cui due o più punti della rotta consecutivi hanno in comune la stessa area di parcheggio più vicina, il secondo caso in cui se la distanza tra due punti di consegna è minore di un valore prefissato (es. 100 m) allora al corriere viene indicata solo l'area DUM più vicina relativa al primo ed il secondo risulta quindi aggregato.
Nello specifico i metodi caratterizzanti sono i seguenti:

1 inorder("coordinata"): replica il funzionamento di inorder() della classe RBTree.java, ma su un grafo

2 searchNeighbors("coordinata"): cerca, iterando per ogni punto del percorso, il gruppo di aree DUM più vicine, sulla base di latitudine o longitudine. Per ogni punto viene inizialmente invocato il metodo searchSplit() che invocherà ricorsivamente searchLat(), restituendo la singola area più vicina; infine, con il metodo getLimit() si ottiene un insieme di aree il cui valore della coordinata non differisce di molto dalla quello dell'area più vicina al punto di consegna

3 searchLat(): per un dato tipo di coordinata, cerca una determinata chiave (non quella esatta ma la più vicina), in questo caso il valore della coordinata, in un albero binario di ricerca; il metodo è stato adattato per una ricerca in un grafo basata su nodi e archi:

```
public static Long searchLat(Long ROOT, double key, Long
    cercanoLat){
   if(ROOT!=null && getLatOfANode(ROOT)==key)
       cercanoLat=ROOT;
   if (ROOT==null || getLatOfANode(ROOT)==key)
       return cercanoLat;
   double distance=Math.abs(getLatOfANode(ROOT)-key);
   if(distance<=MAX_DIST){
       if(cercanoLat!=null){
           if(distance<lessDistance){
```

```
                lessDistance=distance;
                cercanoLat=ROOT;
            }
        }else{
            lessDistance=distance;
             cercanoLat=ROOT;
        }
    }
    if (getLatOfANode(ROOT) > key){
     return searchLat(getLeftChildOfANode(ROOT),
         key,cercanoLat);
    }else{
  return searchLat(getRightChildOfANode(ROOT),
      key,cercanoLat);
    }
}
```

4 getLimit(): per un dato tipo di coordinata, sulla base del nodo n con chiave più vicina restituito dal metodo searchSplit(), cerca un insieme di punti che abbiano un valore per quella coordinata vicino a quello del nodo n, muovendosi a destra ed a sinistra del nodo pivot sulla lista ordinata delle aree DUM

5 searchForPoints(): per ogni punto della rotta interseca le aree trovate in base alla latitudine con quelle trovate in base alla longitudine per trovare eventuali punti in comune che indicano che l'area DUM si trova nel raggio di distanza massima scelto, calcola quindi la distanza euclidea tra le aree trovate ed i rispettivi punti del percorso

6 aggregateBasic(): controlla se due o più punti di consegna hanno in comune il parcheggio più vicino, in questo caso nella lista finale risulteranno aggregati. Partendo dalla lista delle aree DUM, risultante dalle computazioni precedenti quindi ordinata, analizza il parcheggio asso-

ciato ad ogni punto di consegna e lo confronta con quello corrispondente al punto di consegna precedente, se sono uguali li aggrega in un'unica voce della lista

7 aggregateSpecial(): iterando la lista risultante dall'elaborazione del metodo aggregateBasic(), se la distanza tra due punti di consegna associati a due aree DUM consecutive è inferiore a un valore prestabilito di x metri, la prima area DUM o la seconda (in base alla distanza) viene soppressa così che il corriere parcheggi in una sola e da questa effettui entrambe le consegne.

Capitolo 4

Progettazione ed implementazione dell'applicazione mobile

4.1 Panoramica dell'applicazione mobile

L'applicazione è stata creata con lo scopo di facilitare il compito dei corrieri durante i loro tragitti giornalieri di consegna merci, indicando loro una rotta sulla base dei punti in cui è raccomandato parcheggiare per effettuare il carico e lo scarico della mercanzia.
La progettazione e l'implementazione possono essere considerate valide a fini unicamente dimostrativi in quanto viene meno, ad esempio, la reale presenza di un server e le rotte vengono conservate in file .csv, o il calcolo della distanza è approssimato al calcolo euclideo (ciò nonostante sono stati effettuati dei test, incentrati unicamente sulla tipologia di misurazione della distanza, con differenti tool che verranno presentati poi nella sezione "Sviluppi futuri").

4.1.1 Funzionalità

Le funzionalità chiave dell'applicazione mobile sono:

- visualizzazione delle rotte generale e della singola rotta nel dettaglio
- visualizzazione della sagoma di ogni rotta in una mappa
- calcolo e visualizzazione, per ogni rotta, dell'elenco dei parcheggi raccomandati e dei punti di consegna per ogni parcheggio (aggregati)
- visualizzazione dei punti di ogni rotta e dei parcheggi più vicini in una mappa, tramite marker.

4.1.2 Esempio di utilizzo

Nella Figura 4.1 viene mostrato il tipico caso di utilizzo dell'applicazione, la freccia rossa rappresenta l'entry point: all'avvio viene mostrata la schermata home "Your Rutes", nella quale vengono elencate le rotte disponibili e, cliccando sul nome della rotta è possibile passare alla schermata Route Details; mentre, cliccando sull'icona della mappa a destra del nome della rotta verrà visualizzata la schermata Map, nella quale viene disegnata la rotta sulla mappa, attraverso la connessione dei punti di consegna. Cliccando invece sul floating button in alto si riceveranno le nuove rotte create dalla compagnia, e si aggiornerà l'elenco.
Nella schermata Route Details è presente l'elenco dei punti di consegna con un pulsante di switch se si decide di visualizzare uno o più punti nella mappa sottostante e il bottone "Optimize Parking", attraverso il quale si passa alla schermata Parking. Quest'ultima mostra l'elenco ordinato degli indirizzi delle aree DUM, ottenuto sulla base dell'algoritmo applicato alla rotta data. Cliccando su una determinata voce questa verrà espansa e riporterà l'elenco dei punti di consegna che verranno coperti parcheggiando il quell'area; è infine possibile, attraverso il bottone "See all in a map", visualizzare in un'ulteriore schermata Map la mappa contenente i marker corrispondenti alle aree DUM in blu e quelli rappresentanti i punti di consegna in viola.

Figura 4.1: Esempio di utilizzo dell'applicazione

4.2 Ambiente di esecuzione e sviluppo

Il progetto è stato sviluppato in ambiente Android utilizzando l'IDE Android Studio 3.1 ed il linguaggio Java. Il target scelto è stato quello di smartphone e/o tablet in cui la minSdkVersion sia la numero 21, mentre la targetSdkVersion è la 26.
L'applicazione è stata testata su dispositivi reali (Xiaomi Mi5) e virtuali (Nexus 5).
Il dispositivo, dovendo fare uso della localizzazione e della navigazione, deve avere la tecnologia GPS come specificato in AndroidManifest.xml (< uses-feature android:name= "android.hardware.location.gps" />).

4.2.1 IDE Android Studio

Android studio è un ambiente di sviluppo integrato (IDE) per la piattaforma Android, disponibile sotto licenza gratuita Apache 2.0, annunciato da Google nel 2013 e rilasciato per la prima volta nel 2014. Ogni progetto Android Studio contiene uno o più moduli con codici sorgente e file; un modulo può essere di tre tipi: Android app modules, Library modules, Google App Engine modules.
Ogni app module contiene le seguenti cartelle:

- manifest: contiene il file AndroidManifest.xml
- java: contiene i file col codice sorgente Java, incluso il codice dei test JUnit
- res: contiene tutte le risorse che non siano codice, come i layout XML, le stringhe dell'interfaccia utente e le immagini bitmap.

Basato sul software di JetBrains IntelliJ IDEA, Android Studio è stato progettato specificamente per lo sviluppo di applicazioni Android. E' disponibile il download su Windows, Mac OS X e Linux, e sostituisce gli Android Development Tools (ADT) di Eclipse, diventando l'IDE primario di Google per lo sviluppo nativo di applicazioni Android. [21]

4.2.2 Layout e XML

Un layout definisce la struttura per un'interfaccia utente nell'applicazione, come in un'activity. Tutti gli elementi del layout vengono creati utilizzando una gerarchia di oggetti View e ViewGroup; una View rappresenta qualcosa che l'utente può vedere e con cui può interagire, mentre un ViewGroup è un contenitore invisibile che definisce la struttura del layout per le View e altri oggetti ViewGroup.
Gli oggetti View sono solitamente chiamati "widget" e possono essere una delle tante sottoclassi, come Button o TextView. Gli oggetti ViewGroup in genere chiamati "layout" possono essere uno dei vari tipi che forniscono una diversa struttura di layout, come LinearLayout o ConstraintLayout.
Si può dichiarare un layout in due modi:

- Dichiarare gli elementi dell'interfaccia utente in XML. Android offre un semplice vocabolario XML che corrisponde alle classi e sottoclassi View, come quelle per widget e layout. Si può anche utilizzare l'editor di layout di Android Studio per creare il layout XML utilizzando un'interfaccia drag-and-drop.

- Istanziare gli elementi del layout in fase di runtime. L'applicazione può creare oggetti View e ViewGroup (e manipolarne le proprietà) a livello di programmazione.

La dichiarazione dell'interfaccia utente in XML consente di separare la presentazione dell'applicazione dal codice che ne controlla il comportamento.
Utilizzando il vocabolario XML di Android, si possono progettare rapidamente i layout dell'interfaccia utente e gli elementi dello schermo da essi contenuti, nello stesso modo in cui vengono create le pagine Web in HTML, con una serie di elementi nidificati.
Ogni file di layout deve contenere esattamente un elemento radice, che deve essere un oggetto View o ViewGroup. Una volta definito l'elemento radice, è possibile aggiungere oggetti di layout o widget aggiuntivi come elementi

figlio per creare gradualmente una gerarchia di viste che definisce il layout. [22]

4.2.3 Libreria Sparksee mobile

Sparksee è il primo graph database disponibile per applicazioni mobili nei dispositivi Android, disponibile in java e C ++. Per assicurarsi che lo sviluppo dell'applicazione soddisfi i, Sparksee Android richiede Platform 9 (Android 2.3) e Android SDK, e supporta i processori armeabi, armeabi-v7a, x86, mips. [23]

4.3 Gestione e persistenza dei dati

4.3.1 Le rotte

Le rotte vengono salvate nel database interno all'applicazione attraverso la libreria Room che fornisce un livello di astrazione su SQLite (una libreria software scritta in linguaggio C che implementa un DBMS SQL di tipo ACID incorporabile all'interno di applicazioni) per consentire un accesso più affidabile al database sfruttando tutta la potenza di SQLite.
In Room ci sono 3 componenti principali (Figura 4.2):

- Database: contiene il database holder e funge da punto di accesso principale per la connessione sottostante ai dati relazionali persistenti dell'applicazione.

- La classe annotata con @Database dovrebbe soddisfare le seguenti condizioni:

  - Essere una classe astratta che estende RoomDatabase.
  - Includere l'elenco di entità associate al database all'interno dell'annotazione.

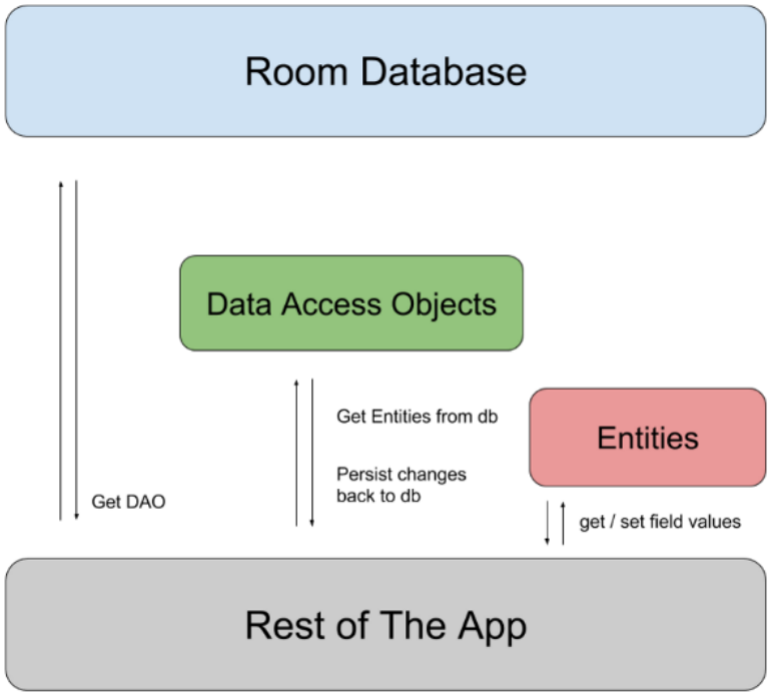

Figura 4.2: Interazioni tra i componenti di Room

  - Contenere un metodo astratto che abbia 0 argomenti e restituisca la classe annotata con @Dao. In fase di runtime, è possibile acquisire un'istanza del Database chiamando Room.databaseBuilder() o Room.inMemoryDatabaseBuilder().

- Entità: rappresenta una tabella all'interno del database.

- DAO: contiene i metodi utilizzati per accedere al database.

L'app utilizza il database Room per ottenere i data access objects, o DAO, associati a quel database. L'applicazione fa poi uso di ogni DAO per ottenere le entità dal database e salvare eventuali modifiche a tali entità nel database. Infine, attraverso un'entity ottiene ed imposta i valori che corrispondono alle colonne della tabella all'interno del database. [24]

Nello specifico, in questa applicazione sono state create due tabelle: routes e delivery_points che contengono le entità di tipo RouteEntity:

```
@Entity(tableName = "routes")
class RouteEntity {
    @PrimaryKey
    @ColumnInfo(name = "id")
    private int id;
    @NonNull
    @ColumnInfo(name = "date")
    @TypeConverters(DateConverter.class)
    private Date date;
```

e quelle di tipo PointEntity:

```
@Entity(tableName = "delivery_points", primaryKeys =
    {"routeId","order"}, foreignKeys = @ForeignKey(entity =
    RouteEntity.class, parentColumns = "id", childColumns =
    "routeId", onDelete = ForeignKey.CASCADE))
class PointEntity {
    @ColumnInfo(name = "routeId")
    private int routeId;
    @ColumnInfo(name = "order")
    private int order;
    @NonNull
    @ColumnInfo(name = "street")
    private String street;
    @ColumnInfo(name = "number")
    private int number;
    @NonNull
    @ColumnInfo(name = "latitude")
    private Double latitude;
    @NonNull
    @ColumnInfo(name = "longitude")
    private Double longitude;
```

La tabella routes contiene l'elenco delle rotte ed ogni rotta ha un id che costituisce la chiave primaria, questo id sarà chiave esterna nella tabella delivery_points, indicando la rotta di appartenenza per ogni punto.
I metodi che vengono messi a disposizione dal DAO consentono di: inserire una rotta, eliminare una o più rotte, ottenere tutte le rotte, inserire un punto, eliminare uno o più punti, ottenere tutti i punti, ottenere tutti i punti relativi ad una specifica rotta.

4.3.2 Le aree DUM

Le aree DUM vengono importate nella memoria interna dell'applicazione, dopo essere state memorizzate in due database a grafo attraverso il procedimento illustrato nel capitolo precedente. Sono quindi da considerarsi informazioni che vengono unicamente lette.
Si è deciso di non memorizzare i parcheggi "ottimizzati" in base alla rotta perché, in uno scenario ottimale, durante il calcolo della rotta ottimizzata si procederebbe a verificare se tali parcheggi siano o possano essere occupati in base al risultato delle statistiche costantemente aggiornate dal comune di Barcellona.

4.4 Implementazione e struttura

Di seguito vengono elencate le principali classi rappresentanti le activity dell'applicazione. Come si può notare dal manifesto dell'applicazione (Figura 4.3), l'activity che viene lanciata per prima all'avvio, mediante l'utilizzo di un intent filter, è HomeActivity.java. Un intent filter dichiara le funzionalità del suo parent component e apre il componente alla ricezione di intent del tipo indicato, mentre filtra quelli che non sono significativi. In questo caso, in particolare, la dicitura android.intent.action.MAIN significa che l'activity è l'entry point dell'applicazione, vale a dire che quando questa si avvia, l'activity viene creata; mentre android.intent.category.LAUNCHER sta ad indicare che l'applicazione sarà visibile nel menù launcher del tele-

fono. Segue il frammento dell'AndroidManifest.xml riguardo l'entry point dell'applicazione:

```
<activity
   android:name=".HomeActivity"
        android:label="@string/app_name">
        <intent-filter>
         <action android:name="android.intent.action.MAIN" />
               <category
                   android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
</activity>
```

4.4.1 HomeActivity.java

Dopo la procedura di login (non implementata) nell'applicazione sarà visibile l'activity Home, se l'app è già stata utilizzata sul dispositivo saranno mostrate, attraverso una RecyclerView, le rotte non completate scaricate in precedenza. Inoltre, attraverso un Floating Action Button è possibile ottenere l'elenco aggiornato delle rotte (se il dispositivo risulta connesso).

RecyclerView

Il widget RecyclerView è da considerarsi come una versione più avanzata e flessibile della ListView, in quanto permette di visualizzare un elenco scorrevole di elementi che si basa su dati che vengono prelevati dinamicamente o in numero consistente. Nel modello RecyclerView, diversi componenti diversi lavorano insieme per visualizzare i dati. Il contenitore generale per l'interfaccia utente è un oggetto RecyclerView si aggiunge al layout. La RecyclerView si riempie con le Views fornite da un gestore di layout prestabilito, si può utilizzare uno di quelli standard (come LinearLayoutManager o GridLayoutManager) o implementarne uno proprio. Le Views nell'elenco

sono rappresentate da oggetti view holder. Questi oggetti sono istanze di una classe definita estendendo RecyclerView.ViewHolder, ogni view holder ha il compito di visualizzare un singolo oggetto mediante una View. La RecyclerView crea solo tanti view holder quanti sono quelli necessari per visualizzare la parte sullo schermo del contenuto dinamico complessivo, ed alcuni in più; mentre l'utente scorre l'elenco, la RecyclerView prende le views fuori dallo schermo e le associa nuovamente ai dati che stanno scorrendo nello schermo. Gli oggetti views holder sono gestiti da un adapter, che viene creato estendendo RecyclerView.Adapter, l'adapter crea i views holder secondo necessità e li lega ai loro dati (binding). Lo fa assegnando il view holder a una posizione e chiamando il metodo onBindViewHolder () dell'adapter. Questo metodo utilizza la posizione del view holder per determinare quale dovrebbe essere il contenuto, in base alla sua posizione di lista.[25]
In questo caso specifico, il layout viene implementato in modo da visualizzare una lista di rotte prelevate dal database dell'applicazione attraverso una chiamata asincrona, etichettate per numero. Gli oggetti presenti all'interno della RecyclerView sono delle CardView che utilizzano un RelativeLayout composto da un'ImageView con un'icona, una TextView con l'identificativo della rotta ed un'ImageButton (con l'immagine di una mappa) attraverso il quale è possibile accedere all'activity RouteMapActivity che contiene la mappa della rotta scelta. Nel caso in cui l'utente faccia click sull'identificativo della rotta, l'adapter, si occupa di riconoscere quale degli elementi è stato premuto e apre l'activity RoutePointsActivity con gli Extra necessari alla visualizzazione dell'itinerario corretto.

4.4.2 RoutePointsActivity.java

Nel layout gestito da quest'activity (Figura 4.3) è nuovamente presente una RecyclerView che contiene la lista dei punti facenti parte della rotta ed ogni voce della lista è stata creata attraverso un RelativeLayout composto da un'ImageView con un'icona, una TextView con l'indirizzo della destinazione ed un pulsante Switch che permette all'utente di aggiungere un marker nella

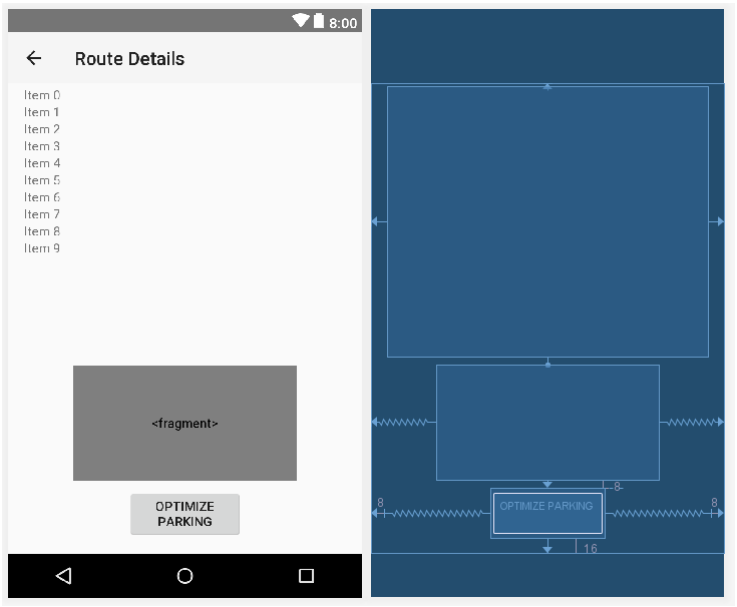

Figura 4.3: Design e blueprint del layout di RoutePointsActivity

mappa sottostante per identificare la posizione del punto di consegna.

Google Maps Android API

Grazie alla Google Maps Android API si possono aggiungere alla propria applicazione delle mappe basate su quelle di GoogleMaps; la libreria gestisce automaticamente l'accesso ai server di Google Maps, il download di dati, la visualizzazione e i riscontri alle gesture dell'utente sul touch screen. E' inoltre possibile aggiungere informazioni aggiuntive tramite markers, polygons e overlays ad una mappa basica e restringere la vista dell'utente ad un'area specifica nella mappa. Per inserire una mappa in una schermata è opportuno:

1 ottenere un' API Key di Google Maps e aggiungere gli attributi necessari nel manifest:
< meta-data android:name="com.google.android.geo.API_KEY"
android:value="@string/google_maps_key" / >

2 aggiungere un oggetto Fragment all'Activity che gestirà la mappa

3 implementare l'interfaccia OnMapReadyCallback ed utilizzare il metodo callback onMapReady(GoogleMap) per gestire l'oggetto GoogleMap; l'oggetto è la rappresentazione interna della mappa stessa, per impostare le opzioni di visualizzazione per una mappa, si può modificare il suo oggetto GoogleMap

4 invocare getMapAsync() sul fragment per registrare la callback.[26]

Infine, attraverso un Button l'utente può richiedere che si passi alla schermata che riporta l'elenco dei parcheggi ottimali utilizzando un Intent che contenga negli Extra una lista dei punti della rotta in oggetto, questo perchè così l'activity successiva non dovrà nuovamente richiedere al database Room l'elenco dei punti della rotta.
Il funzionamento di un Intent:

```
Intent intent = new Intent(this, ParkingPointsActivity.class);
intent.putExtra("routeId", "RouteId1");
intent.putParcelableArrayListExtra("pointOfTheRoute",routePoints);
startActivity(intent);
```

Gli extras sono tra gli attributi addizionali che possono essere inclusi in un Intent, rappresentano informazioni aggiuntive e possono essere usati dal component ricevente.
Dovendo, in questo caso, comunicare l'elenco dei punti della rotta sotto forma di ArrayList, il metodo utilizzato è stato public Intent putParcelableArrayListExtra (String name, ArrayList value) perchè il valore passato deve estendere l'interfaccia Parcelable che permette da serializzazione e la deserializzazione di oggetti Java utilizzati in Android.
L'interfaccia Parcelable descrive una modalità di registrazione di un oggetto con tutti i suoi dati primitivi come int, boolean, String - oppure un qualsiasi oggetto che a sua volta implementa Parcelable - in un oggetto Parcel associato ad esso.
Per rendere gli oggetti di tipo Point parcellizzabili è stato utilizzato il suppor-

to di http://www.parcelabler.com che, dato un tipo di oggetto, implementa in maniera automatica su di esso la classe Parcelable.

4.4.3 RouteMapActivity.java

Attraverso questa activity l'utente può visualizzare su una mappa una linea che connette in linea d'aria i punti di consegna tra loro, rispettando l'ordine prefissato; è stata pensata per premettere all'utente di avere un'idea generale della zona che dovrà coprire.
La linea è in realtà una Polyline, ovvero un elenco di punti in cui vengono tracciati segmenti lineari tra punti consecutivi; questo tipo di oggetto viene creato aggiungendolo alla mappa GoogleMaps in questo modo:
Polyline polyline = mMap.addPolyline(new PolylineOptions().clickable(true) .addAll(routeToMap));
dove routeToMap è un ArrayList contenente l'elenco delle coordinate dei punti.

4.4.4 ParkingPointsActivity.java

Da RoutePointsActivity l'utente può decidere di passare alla visualizzazione, in questa schermata, delle aree DUM in cui parcheggiare.
Direttamente nel metodo onCreate() vengono estratti, attraverso l'intent, i punti della rotta: ArrayList<Point> points = intent.getParcelableArrayList Extra("pointOfTheRoute"); e su questi viene invocato l'algoritmo presentato nel capitolo 3: ArrayList<ParkDetailed> optimized = Algorithm.compute (points) ed ogni elemento della lista viene poi inserito in un'ExpandableList-View.
Infine, per mezzo del bottone "See all in a map" si passa all'activity Dum-MapActivity che mostra come aree DUM e punti di consegna sono dislocati nella mappa.

ExpandableListView

È una vista che mostra gli elementi in un elenco a due livelli a scorrimento verticale; per questo differisce dalla ListView consentendo due livelli: gruppi che possono essere espansi individualmente per mostrare i propri "children". Gli elementi provengono da una classe che implementa un ExpandableListAdapter associato alla view, che inserisce i dati nelle posizioni corrette; l'implementazione di questa interfaccia fornisce l'accesso ai dati dei children (suddivisi per gruppi) e crea anche istanze di views per i children ed i gruppi. Le liste espandibili possono mostrare un indicatore accanto a ciascun elemento per visualizzare lo stato corrente dell'oggetto (lo stato è generalmente uno tra gruppo espanso, gruppo compresso, child o last child)[27].
In questo caso, l'oggetto ParkDetailed ha le seguenti variabili di istanza:

```
private Long dumIdentifier;
private Double latitude;
private Double longitude;
private String address;
private ArrayList<Point> deliveries
```

La variabile dumIdentifier contiene l'identificativo del nodo DUM nel grafo, latitude, longitude e address le coordinate e l'indirizzo dell'area DUM, ed infine deliveries è la lista dei punti di consegna attribuibili (mediante le varie computazioni) a quella determinata zona di parcheggio. Nella lista risulta quindi che il valore della variabile address degli oggetti ParkDetailed va ad identificare i gruppi, mentre il valore della variabile address degli oggetti Point occupa il posto di child; ogni voce dell'ExpandableListView è composta da due LinearLayout, uno per il gruppo ed un altro per ogni singola voce del gruppo (Figura 4.4).

Figura 4.4: Oggetti in stato compresso ed espanso

4.4.5 DumMapActivity.java

Questa activity è molto simile a RouteMapActivity, è infatti composta da una mappa sulla quale è stata aggiunta una polyline; ma questa volta i punti connessi rappresentano le aree DUM.
Sono inoltre presenti dei marker di due colori differenti: azzurro per le aree DUM e viola per i punti di consegna, questo in modo tale da facilitare l'individuazione e la navigazione verso tali luoghi. Cliccando sui marker appare l'indirizzo esatto del punto di interesse ed è possibile richiedere le indicazioni stradali selezionando l'icona della freccia e passare all'applicazione Google Maps (Figura 4.5).

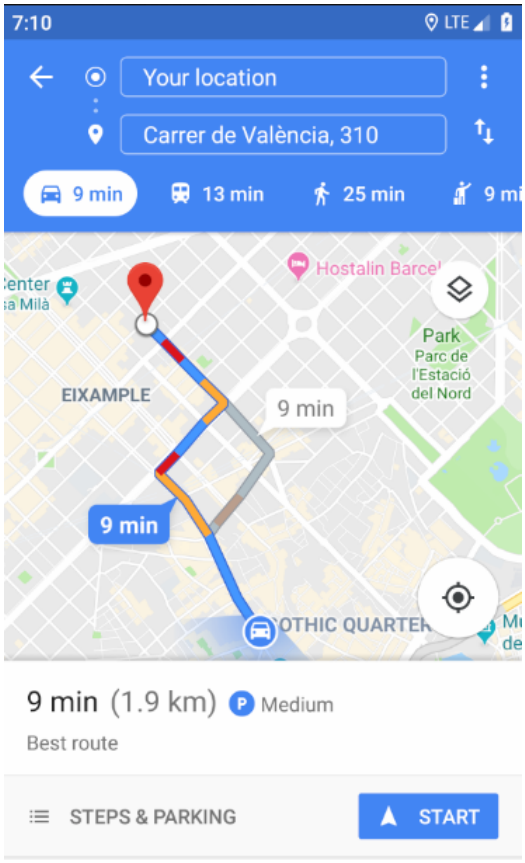

Figura 4.5: Navigazione con Google Maps

Capitolo 5

Conclusioni

Questo progetto ha tentato di fornire una proposta di applicazione mobile per i corrieri operanti nella città di Barcellona, con l'obiettivo di cercare di ridurre i disagi del singolo, dovuti alla perdita di tempo per la ricerca di un'area DUM in cui parcheggiare, e della comunità, che deve convivere con traffico ed emissioni nocive crescenti. E' bene considerare, infatti, che l'attuazione di una politica per un gruppo di cittadini può influire anche su individui non appartenenti al gruppo target. Vuole inoltre essere uno spunto di riflessione nell'ambito delle possibili idee di tecnologie da adottare per rendere una città più "smart" e si collocherebbe al livello più alto del modello IT architetturale della città, previsto nell'"Urban Platform". E' stato inizialmente valutato lo scenario in cui si sarebbe andato ad operare: altre applicazioni già operative che si occupano di ottimizzare rotte (ed i loro punti deboli), o che forniscono un aiuto nella ricerca del parcheggio, e le caratteristiche del territorio e della città di Barcellona (la presenza delle aree DUM, la percorribilità delle strade, ecc.).
Partendo da questo punto si è ideato un algoritmo, che però non si è rivelato ottimale dal punto di vista della complessità computazionale, perchè calcola la distanza da ogni area DUM ad ogni punto della rotta assegnata (dovendo poi scartare la maggior parte delle soluzioni perchè si rivelano irrilevanti, ovvero ad una distanza maggiore di quella minima prefissata). Si è poi in-

tuito che la chiave per risparmiare importanti risorse fosse restringere, già in partenza, il range di aree DUM da cui calcolare la distanza. E' stata quindi perfezionata la prima soluzione, strutturando i dati come alberi Red-Black per ridurre le tempistiche di ricerca ed evitare di considerare dati non rilevanti. Entrambe le soluzioni fanno uso del database a grafo Sparksee per elaborare le aree DUM più vicine ai punti del percorso.
Infine è stata creata l'applicazione Android, che utilizza l'algoritmo creato in precedenza, per dare modo al corriere (utente target dell'applicazione) di poter scaricare le rotte che gli vengono assegnate, visualizzarle sulla mappa e ricevere suggerimenti che gli permettono di localizzare le zone di parcheggio riservate più facilmente. Nell'applicazione non sono state implementate tecniche di autenticazione dell'utente e di richiesta del percorso perché già esistenti nelle app analizzate in precedenza.

5.1 Sviluppi Futuri

L'applicazione di cui si è descritto lo sviluppo rappresenta però la base per un progetto più ampio ed avanzato dal punto di vista della qualità delle prestazioni. Vengono di seguito presentate alcune idee per future aggiunte e migliorie più o meno prioritarie:

- Introduzione di mappe disponibili offline (es. attraverso la funzione "Esporta" di OpenStreetMap), per una duplice funzione: visualizzazione delle porzioni di mappa e calcolo delle distanze senza dover aspettare i tempi di risposta di molteplici richieste http

- Utilizzo del Military Grid Reference System (MGRS) per la localizzazione delle aree DUM più vicine ai punti di consegna. Un riferimento alla griglia MGRS è un sistema di riferimento a punti. Quando viene utilizzato il termine 'grid square', ci si può riferire ad un quadrato con un lato di lunghezza di 10 km (6 miglia), 1 km, 100 m (328 piedi), 10 m

o 1 m, a seconda della precisione del coordinate fornite. Un esempio di una coordinata MGRS, o riferimento di griglia, sarebbe 4QFJ12345678, che consiste di tre parti: 4Q (designazione della zona griglia, GZD), FJ (l'identificatore quadrato di 100.000 metri), 12345678 (posizione numerica; est è 1234 e north è 5678, in questo caso specificando una posizione con una risoluzione di 10 m). Il numero di cifre nella posizione numerica deve essere pari: 0, 2, 4, 6, 8 o 10, a seconda della precisione desiderata. Quando si modificano i livelli di precisione, è importante troncare anziché arrotondare i valori di est e nord per garantire che il poligono più preciso rimanga entro i limiti del poligono meno preciso. In relazione a ciò, l'angolo sud-ovest del poligono è il punto di etichettatura per un intero poligono. [28]

- Integrazione con la piattaforma già esistente Cigo!Fleet: sistema di autenticazione per l'applicazione mobile e di richiesta dei dati sulle rotte al server aziendale

- Creazione di un sistema globale per la raccolta dei dati provenienti dai singoli corrieri che fanno uso dell'applicazione (es. aree DUM maggiormente utilizzate) col fine di elaborarli (ad esempio mediante statistiche) per creare informazioni che possano essere impiegate per rendere l'algoritmo per la ricerca dei parcheggi il più affidabile possibile

- Utilizzo delle tecnologie Beacon Bluetooth per aggiornamenti in tempo reale sullo stato (occupato o libero) dell'area DUM; in modo tale da permettere ai corrieri una verifica in tempo reale sulle aree di sosta future (es. prima di partire per il punto di consegna successivo il corriere può rieseguire l'algoritmo che, se connesso alla rete, potrà consigliare una zona di sosta "probabilmente" libera)

- Possibilità per l'utente di includere nell'elenco delle zone di parcheggio possibili anche quelle a pagamento (nel caso di un notevole risparmio in termini di tempo)

- Perfezionamento del design e dell'esperienza di utilizzo dell'applicazione, migliorando User Interface ed User Experience.

Bibliografia

[1] Santos, A. C., Duhamel, C. and Urrutia, S. (2019), Special issue on " Developments in Optimization for Smart Cities " . Intl. Trans. in Op. Res., 26: 379-380. doi:10.1111/itor.12600

[2] STARICCO, Luca. Smart Mobility Opportunities and Conditions. Tema. Journal of Land Use, Mobility and Environment, v. 6, n. 3, p. 342-354, nov. 2013. ISSN 1970-9870.

[3] https://www.quora.com/Which-is-the-best-delivery-driver-route-planner-app

[4] http://www.diariosigloxxi.com/texto-diario/mostrar/951378/cuanto-tiempo-pierde-buscando-aparcamiento

[5] https://downloads.conduent.com/content/usa/en/ebook/european-urban-transportation-survey.pdf

[6] https://www.telegraph.co.uk/motoring/news/10082461/Motorists-spend-106-days-looking-for-parking-spots.html

[7] DUDHANE, Nilima A.; PITAMBARE, Sanjeevkumar T. Location based and contextual services using bluetooth beacons: New way to enhance customer experience. Lecture Notes on Information Theory Vol, 2015, 3.1.

[8] https://parclick.it/

[9] https://www.parkunload.com/

[10] https://www.parkopedia.it/

[11] http://www.smart-cigo.com/smart-cigo/

[12] https://www.roadwarriorllc.com/

[13] Mehdi Nourinejad, Adam Wenneman, Khandker Nurul Habib, Matthew J. Roorda, Truck parking in urban areas: Application of choice modelling within traffic microsimulation, Transportation Research Part A: Policy and Practice,Volume 64, 2014,Pages 54-64,ISSN 0965-8564,https://doi.org/10.1016/j.tra.2014.03.006.

[14] Bakici, T., Almirall, E. and Wareham, J. J Knowl Econ (2013) https://doi.org/10.1007/s13132-012-0084-9

[15] https://www.c40.org/case_studies/barcelona-s-smart-city-strategy

[16] ANGLES, Renzo; GUTIERREZ, Claudio. An introduction to Graph Data Management. In: Graph Data Management. Springer, Cham, 2018. p. 1-32.

[17] Jeff Carpenter, Why you should use a graph database, https://www.infoworld.com/article/3251829/nosql/why-you-should-use-a-graph-database.html .

[18] Damaris Coll, Sparksee Overview, https://www.slideshare.net/SparsityTechnologies/sparksee-overview

[19] c. Thomas Cormen, Charles E. Leiserson, Ronald Rivest, Introduction, in Introduction to Algorithms, 2Âª ed., Cambridge, Massachusetts, The MIT Press, 1998 .

[20] Red-Black Trees, http://pages.cs.wisc.edu/ paton/readings/Red-Black-Trees/

[21] https://developer.android.com/studio/intro

[22] https://developer.android.com/guide/topics/ui/declaring-layout

[23] http://sparsity-technologies.com/blog/how-to-install-configure-sparksee-mobile-for-android/

[24] https://developer.android.com/training/data-storage/room/

[25] https://developer.android.com/guide/topics/ui/layout/recyclerview

[26] https://developers.google.com/maps/documentation/android-sdk/map

[27] https://developer.android.com/reference/android/widget/ExpandableListView

[28] http://earth-info.nga.mil/GandG/coordsys/grids/universal_grid_system.html