A2A Challenge
Technical Report

De Giosa Matteo,
Pagliara Gianluca,
Santoro Andrea

CdLM Data Science
Università degli Studi di Milano-Bicocca

Indice

Elenco delle figure

Capitolo 1

Introduzione

La corretta gestione dei rifiuti è una delle sfide emergenti in tutto il mondo, a causa della sua diretta relazione con l'ambiente: l'urbanizzazione dilagante, così come le comuni attività giornaliere, producono una grandissima quantità di rifiuti, su tutti i livelli (residenziale, commerciale, industriale). Tra tutti gli step della gestione dei rifiuti, la fase di raccolta presenta alcune delle maggiori criticità; nel processo "tipico" di raccolta, infatti, i veicoli partono da un deposito comune e visitano tutte le posizioni disposte su un percorso fisso: questo approccio in genere porta ad uno spreco di risorse - tempo, denaro, emissioni - dovuto alla visita di cestini che ancora non necessitano di essere svuotati. Inoltre, in assenza di misurazioni in real-time, un cestino potrebbe non venire svuotato anche se diventa pieno prima della raccolta programmata. Per questo motivo, un sistema di raccolta efficiente deve avere a disposizione i dati di riempimento dei cestini in real time: in questo modo un algoritmo può costruire un percorso di raccolta ottimizzato in base ai cestini da svuotare.

La progettazione di tali algoritmi, tuttavia, presenta non poche criticità, dovute perlopiù alla complessità computazionale del problema: il Vehicle Routing Problem è infatti uno dei più noti problemi NP-hard.

Obiettivo di questo articolo è dunque quello di presentare un possibile approccio al processo di raccolta dei cestini e ottimizzazione dei percorsi, a partire da dati reali. In particolare, proporremo una soluzione al problema basata su un approccio "cluster first, route second" che possa rivaleggiare con le tecniche state-of-the-art disponibili al momento sia in termini di prestazioni che di robustezza e scalabilità.

Veranno descritte tutte le componenti di questa soluzione, per poi testarla ampiamente anche a confronto con altre: in particolare, sarà interessante il confronto con l'approccio di Google senza clustering, per mezzo della libreria open-source OR-Tools.

La struttura dell'articolo è dunque la seguente:

- Il capitolo 2 presenta le nozioni preliminari e gli approcci più diffusi in letteratura nella risoluzione del problema, concentrandosi in particolare sull'approccio "Cluster-First Route-Second", oggetto di questo lavoro;

- Il capitolo 3 descrive il dataset a disposizione, riassume le operazioni di preprocessing e pulizia dei dati, e riporta la formulazione matematica del problema (Capacitated Vehicle Routing Problem, CVRP) con i vincoli da rispettare;

- Il capitolo 4 presenta l'approccio usato in questo articolo per il clustering dei cestini;
- Il capitolo 5 presenta l'algoritmo di routing implementato;
- Il capitolo 6 espone invece il metodo di valutazione del sistema, e lo confronta con altri approcci popolari e non;
- Il capitolo 7 trarrà le conclusioni dal lavoro svolto, con qualche cenno agli sviluppi futuri.

Capitolo 2

Nozioni preliminari

Nella ricerca scientifica il problema della raccolta dei rifiuti viene formulato sotto forma di vehicle-routing problem (VRP) [1], una classe di problemi nell'ambito della ricerca operativa, molto simile in natura al traveling salesman problem (TSP): si tratta di progettare dei percorsi di consegna dal costo minimo che partano dallo stesso punto (magazzino o "depot") e che servano un gruppo di clienti distribuito geograficamente. E' uno dei più famosi problemi NP-hard, ovvero particolarmente oneroso dal punto di vista computazionale.

Si tratta poi nel caso in esame di una sottocategoria di questo problema, data la presenza di un vincolo sulla capacità dei veicoli di raccolta (capacitated VRP, o CVRP). Diversamente dai problemi di VRP di qualche decina di anni fa, poi, abbiamo anche a disposizione delle tecnologie di smart bin che ci forniscono i dati in real-time sul riempimento dei cestini.

In letteratura si possono distinguere tre diversi approcci alla risoluzione del CVRP: approccio convenzionale, euristico e meta-euristico.

Nell'approccio convenzionale vengono in genere usate tecniche di programmazione matematica, come la programmazione lineare o problemi lineari misto-interi (mixed integer linear programming, o MILP) [7]; tecniche stardard di ricerca operativa quindi, come il Branch & Bound [19] o il Branch & Cut [22], possono essere utilizzate per risolvere problemi di CVRP con un numero di nodi basso, dato che analizzano tutte le soluzioni possibili. Il più grande problema di CVRP risolto con queste tecniche, a nostra conoscenza, aveva 135 nodi [24].

A partire dal 1964 con l'algoritmo di Clarke and Wright [6] sono stati proposti diversi approcci "euristici", che producono cioè buone soluzioni esplorando un sottoinsieme dello spazio di ricerca. Possiamo individuare quindi tre sottocategorie di soluzioni euristiche: euristiche costruttive, euristiche a due fasi ed euristiche migliorative.

Le euristiche costruttive, di cui fa parte l'algoritmo di Clarke and Wright, costruiscono gradualmente una soluzione accettabile partendo da zero, cercando di minimizzare il costo totale.

Le euristiche a due fasi, d'altra parte, scompongono il problema in due parti: clustering (raggruppamento) dei nodi in gruppi e costruzione del percorso per ogni nodo (routing); ad ogni cluster viene dunque assegnato un veicolo, per cui i sottoproblemi da risolvere prendono la forma di "Traveling Salesman Problem", da risolvere con algoritmi esatti o con soluzioni note che garantiscono un grado di approssimazione massimo, come l'algoritmo di Christofides [4]. L'euristica a due fasi può essere

formulata sia nella variante "Cluster-First, Route Second", le cui prime implementazioni [13] [11] risalgono agli anni settanta-ottanta, sia nella variante "Route-First, Cluster-Second", poco competitiva rispetto agli altri approcci.

Le euristiche migliorative, infine, si pongono l'obiettivo di migliorare le soluzioni trovate eseguendo una serie di scambi di vertici all'interno dei percorsi trovati o tra i percorsi di due veicoli diversi. Un esempio di tali euristiche è l'algoritmo 2-opt, presentato da Croes [8] nel 1958: l'idea dietro l'algoritmo è di riordinare un percorso che interseca se stesso eliminando l'intersezione.

Negli ultimi anni sono tuttavia emersi i cosiddetti approcci "meta-euristici", che sfruttano strategie di ricerca globale per l'esplorazione dello spazio di ricerca. Il termine "metaheuristic", citato per la prima volta da Glover nel 1986 [14], viene definito dallo stesso come "master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality" [15]. Gli approcci meta-euristici incorporano in genere una componente stocastica, e sono in genere algoritmi nature-inspired, ovvero sono progettati come imitazione di comportamenti animali o processi evolutivi. L'approccio meta-euristico è diventato molto popolare negli ultimi anni nella risoluzione del VRP: citiamo ad esempio l'applicazione dell'algoritmo delle colonie di formiche (ant colony optimization) [20] - applicato poi tra l'altro anche ad un caso reale di raccolta rifiuti in una città del Ghana [23] -, l'algoritmo genetico [25], la tabu search [12] o la particle swarm optimization (ispirata al movimento degli sciami) [17].

Non sono rare però in letteratura le voci controtendenza, a favore di approcci con clustering: in [26] l'autore, pur utilizzando un approccio meta-euristico (ant colony), suggerisce di clusterizzare i nodi (con il k-means) quando questi formano "naturalmente" dei cluster, per velocizzare l'esecuzione del sistema, anche a discapito della precisione. Tuttavia, in fase di valutazione, trova che operare il clustering nei network ben strutturati non solo velocizza l'esecuzione, ma migliora anche il risultato finale; anche in [27] gli autori giungono alla conclusione che, per problemi di larga scala, è buona pratica trovare dei cluster di nodi (con un k-means) e per ogni cluster trovare il percorso meno costoso attraverso delle tecniche meta-euristiche, nel caso specifico un ant-colony seguito dall'algoritmo migliorativo 2-opt.

Ispirandoci a questi lavori la soluzione da noi proposta sarà un approccio cluster-first, route-second: dopo diversi test infatti siamo giunti alla conclusione che un clustering intelligente, unito ad un algoritmo di routing per il TSP che garantisca un grado di approssimazione massima, possa fare anche meglio di un approccio meta-euristico "innovativo".

Capitolo 3

Preprocessing e formulazione del problema

Il dataset fornito da A2A consta di 814.727 osservazioni: ognuna di esse rappresenta una rilevazione effettuata da un cestino, nell'arco temporale che va dal al *01/09/2019* al *07/10/2019*.

Di seguito elenchiamo le variabili, o colonne, più rilevanti al suo interno:

- *Bin serial*: l'identificativo univoco di ogni cestino;
- *Bin level*: livello di riempimento di ogni cestino, valore discreto da 1 a 4;
- *Occluded*: variabile che indica se un cestino è ostruito o meno;
- *Detected at*: data e ora a cui si riferisce il valore del bin level;
- *Esterno*: variabile che indica se il cestino è esterno o meno;
- *Tipo anomalia*: la tipologia di anomalia eventualmente riscontrata;
- *Latitudine*;
- *Longitudine*;

Seguono poi una serie di variabili nominali sulla posizione del cestino (via, cap, area gestionale, etc.) superflue data la presenza delle coordinate GPS, e una serie di variabili temporali (data installazione, prima trasmissione, etc.) che non sono rilevanti ai fini del problema.

Le colonne utili alla costruzione del modello e alla sua valutazione, in ultima istanza, sono le seguenti: bin serial, bin level, detected at, latitudine e longitudine.

Possiamo banalmente eliminare la colonna "Occluded" trasformando ogni cestino ostruito in un cestino "pieno", ovvero con bin level 4: un cestino ostruito richiede in ogni caso un intervento manuale prioritario, alla pari di un cestino pieno.

Analizzando la colonna "Esterno", notiamo poi che solo 2 cestini sono interni: questa informazione tuttavia ci è poco utile, dato che non viene fornita nessuna indicazione alternativa su come trattare questi cestini. Il nostro modello dunque non farà distinzione tra cestini esterni o interni.

Passiamo poi alla colonna "Tipo anomalia": notiamo che circa il 10% dei dati ha qualche tipologia di anomalia, mentre il resto delle righe riporta un valore nullo

per questa colonna; in questo caso, data l'assenza di informazioni sulla natura di questa colonna, decidiamo di rimuovere dal dataset le osservazioni che presentano delle anomalie.

Il dataset ripulito consta di 799.159 righe, ma non sarà lo stesso dato in pasto ai modelli di apprendimento: verrà poi riprocessato nella sezione 6 per generare le finestre corrispondenti ai turni di raccolta dei cestini, che ci serviranno per valutare la bontà del sistema proposto.

3.1 Formulazione del problema

Il CVRP nell'ambito della raccolta dei rifiuti è definito come il processo di visita di tutti i nodi (cestini) da una flotta di veicoli, con il vincolo che tutti i veicoli partano da un deposito comune e ci facciano ritorno. Il sistema, che ha a disposizione le coordinate e il livello di riempimento dei cestini, ha l'obiettivo di generare dei percorsi per i veicoli che minimizzino la funzione di distanza o di costo, rispettando allo stesso tempo i seguenti vincoli:

- Tutti i veicoli partono dallo stesso deposito e ci fanno ritorno;
- Un cestino è visitato da un solo veicolo per volta;
- La capacità di ogni veicolo, in termini di numero massimo di cestini svuotabili C, non può essere superata;
- E' necessario massimizzare la vuotatura dei cestini quando sono pieni, e minimizzarla quando sono vuoti o mezzi pieni.
- Tutti i cestini devono essere svuotati almeno una volta al giorno.
- Bisogna dichiarare in anticipo l'ora di svuotamento di ogni cestino, per eventuali ispezioni.

Il terzo vincolo è stato tenuto in conto nella fase di clustering del problema, andando a bilanciare i cluster creati; il quarto invece introducendo la nozione di Threshold Waste Level (TWL), una variabile che definisce il livello di riempimento per cui i cestini sono considerati "da svuotare". Il quinto, infine, è stato rispettato aggiungendo all'ultimo turno i cestini mai riempiti e mai svuotati durante la giornata.

Esistono diverse formulazioni matematiche del CVRP; riportiamo di seguito quella di Christofides [5]: X_{ijm} indica se il veicolo m può arrivare dal cestino i al cestino j, N indica il numero di cestini da svuotare e M il numero di veicoli a disposizione.

$$X_{ijm} = \begin{cases} 1, & \text{se il veicolo } m \text{ può viaggiare da } i \text{ a } j \\ 0, & \text{altrimenti} \end{cases} \tag{3.1}$$

L'obiettivo è dunque minimizzare la funzione di distanza Z, rispettando i vincoli precedentemente elencati:

$$Z = min \sum_{i=0}^{N} \sum_{j=0}^{N} d_{ij} \sum_{m=1}^{M} X_{ijm}$$

con d_{ij} che rappresenta il costo per andare da i a j, ovvero la distanza tra i due punti.

Capitolo 4

Clustering

Sebbene il clustering possa sembrare una soluzione obsoleta in uno scenario dominato da algoritmi evolutivi e memetici, in un caso reale, in cui ci sono chiari vincoli di efficienza computazionale e tempestività delle risposte, esso rimane un'opzione viabile per la semplificazione del problema del CVRP. Per il sistema progettato abbiamo preso come base l'algoritmo k-means, operando però delle modifiche sostanziali al modello di clustering:

- Modifica della funzione di distanza, in modo da creare cluster della forma desiderata;
- Bilanciamento dei nodi all'interno dei cluster, in modo da rispettare il vincolo sulla capacità dei veicoli.

Il k-means è uno dei più famosi modelli di clustering [18]. L'algoritmo segue di base una procedura iterativa, con l'obiettivo di minimizzare la varianza all'interno dei cluster creati: inizialmente vengono create k partizioni (numero di cluster) randomicamente o con determinate euristiche; per ogni partizione si calcola un centroide, e tutti i nodi vengono assegnati alla partizione con centroide più vicino; ad ogni iterazione vengono dunque ricalcolati i centroidi e riassegnati i nodi, finchè l'algoritmo non converge, ovvero i centroidi non cambiano rispetto all'iterazione precedente. Viene usata la misura di distanza euclidea per calcolare la distanza tra due nodi.

Tuttavia un semplice k-means non è adatto al problema affrontato, in quanto tende a formare cluster di forma globulare: idealmente vorremmo dei cluster longilinei, disposti a raggiera intorno al magazzino. Per fare questo dobbiamo operare sulla funzione di distanza usata dall'algoritmo. Ispirandoci al lavoro svolto da [3], dunque, definiamo la misura di distanza tra due nodi i e j come

$$\delta_{ij} = \alpha\theta_{ij} + \beta\rho_{ij}$$

dove θ_{ij} è la distanza angolare tra i due nodi, ovvero l'angolo che formano con il magazzino, considerato come origine; ρ_{ij} è la distanza spaziale tra i due nodi; α e β i rispettivi pesi delle due distanze, fissati in modo che $\alpha + \beta = 1$.

In questo modo un valore più alto di α renderà i cluster più allungati e disposti a raggiera attorno al magazzino, mentre un valore alto di β li renderà di forma più globulare, a distanze progressivamente maggiori dal magazzino. La figura 4.1 mostra questo comportamento.

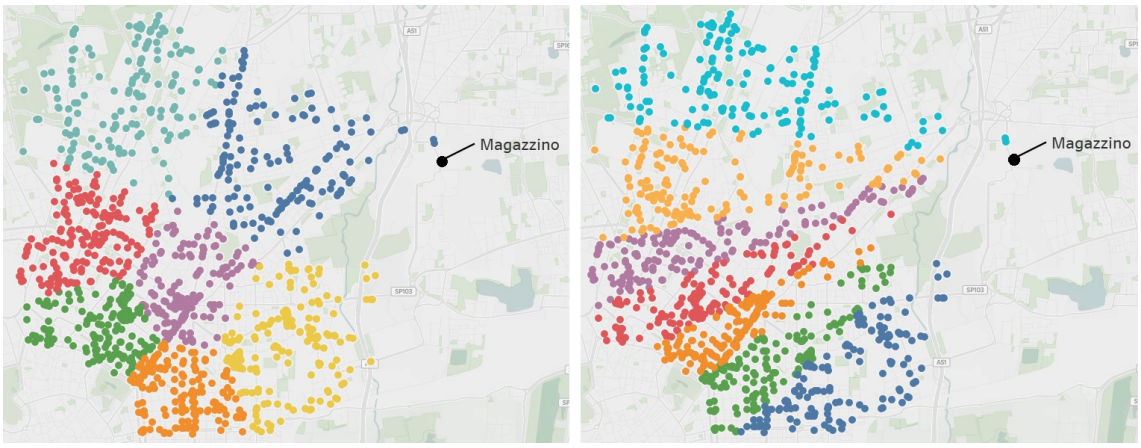

Figura 4.1: A sinistra i cluster ottenuti con $\alpha = 0$ e $\beta = 1$, a destra i cluster ottenuti con $\alpha = 1$ e $\beta = 0$.

Elemento fondamentale per il clustering è poi la scelta del numero di cluster, k. La soluzione adottata è semplice, e ricalca quanto fatto dagli autori in [26], ovvero dividere il numero totale dei cestini da svuotare per la capacità di ogni veicolo: in questo modo saranno schierati in totale k veicoli, uno per ogni cluster.

A questo punto però bisogna assicurarsi che il numero di nodi in ogni cluster non superi la capacità dei veicoli. In un precedente lavoro, è stato introdotto il *Constrained K-Means* [2] che permette di realizzare cluster con una dimensione minima fissata. Un approccio equivalente può essere usato per imporre il vincolo sulla dimensione massima. Inoltre, imponendo la dimensione minima uguale a $|cestini|/|cluster|$ si possono anche ottenere cluster pressocché bilanciati nel numero di cestini.

Il problema del clustering con vincoli viene risolto attraverso un problema di ottimizzazione lineare [1]. Come per un classico k-means, si inizializzano casualmente i centroidi e si itera associando ogni record al centroide più vicino, aggiornando di volta in volta i centroidi che rappresentano ciascun cluster, finché l'algoritmo non converge (i centroidi rimangono uguali per due iterazioni consecutive). Definiamo quindi un sottoproblema di assegnazione che verrà eseguito da un solver MIP (*Mixed-Integer Programming*). Il sottoproblema è così definito:

- variabili:
  - matrice delle assegnazioni: ogni nodo, per ogni cluster, deve essere associato ad un valore in $\{0, 1\}$ (1=nodo assegnato a quel cluster);
  - vettore degli outflow: per ogni cluster, l'outflow deve essere compreso fra 0 e il numero di nodi meno la dimensione minima;
- funzione obiettivo da minimizzare: classica funzione obiettivo del k-Means, in cui si minimizza il quadrato della norma euclidea della distanza di ogni punto dal centroide più vicino;
- vincoli:
  - ogni nodo deve essere assegnato ad uno ed un solo cluster;

[1] Idea suggerita dalla seguente implementazione: https://github.com/Behrouz-Babaki/MinSizeKmeans/blob/master/README.md

- per ogni cluster, l'outflow deve essere uguale al numero di nodi assegnati a quel cluster meno la dimensione minima;
- per ogni cluster, l'outflow deve essere minore o uguale alla dimensione massima meno la dimensione minima;
- la somma degli outflow dei vari cluster deve essere uguale al numero di nodi meno la dimensione minima moltiplicata per k.

Per risolvere il sottoproblema si è fatto uso della libreria Python PuLP, con il MIP solver open-source CBC. Esistono altri MIP solvers in commercio più efficienti, ma che richiedono una licenza di utilizzo, tuttavia anche con CBC i tempi sono abbastanza buoni, come vedremo nel Capitolo 6.

Chiameremo la soluzione adottata Shape-Controlled Clustering.

Capitolo 5

Percorso minimo

Determinati i cluster, bisogna trovare, per ciascuno di essi, il percorso minimo che attraversi tutti i nodi del cluster una sola volta, partendo da un punto (il deposito) e tornando nello stesso. Questo problema è definito Traveling Salesman Problem, o TSP, e per risolverlo si possono adottare algoritmi esatti (nel caso in cui i nodi siano pochi) o euristiche, similmente a quanto accade per il VRP, che è la generalizzazione del TSP.

La soluzione più diretta al problema sarebbe quella di provare tutte le permutazioni possibili dei nodi, $n!$, ma la complessità computazionale sarebbe ingestibile già per 20 nodi. Esistono poi numerosi algoritmi esatti basati su tecniche di programmazione dinamica o lineare, ma la complessità minima raggiunta da questi approcci è di $O(2^n)$, comunque esponenziale.

Per questa motivazione siamo costretti ad affidarci a delle euristiche, in particolar modo l'algoritmo di Christofides [4], che nella sua formulazione originale garantisce, nel caso pessimo, che il percorso generato sia inferiore a 1.5 volte quello ottimale.

In particolare ci affidiamo ad un'implementazione dell'algoritmo offerta dal software di Google OR-Tools [1], una libreria open-source messa a disposizione dall'azienda per affrontare problemi di ottimizzazione combinatoria come il routing dei veicoli.

La libreria, diventata popolare negli ultimi anni per aver vinto diverse medaglie in competizioni di constraint programming (programmazione a vincoli) [21], fa uso di tecniche e di solver che sono alla base degli stessi servizi offerti dalla società di Mountain View, come Google Maps.

L'implementazione dell'algoritmo di Christofides all'interno della libreria, in particolare, non rispetta il vincolo del caso pessimo proposto originariamente dall'autore, ma compensa ciò con tempi d'esecuzione di diversi ordini di grandezza più brevi.

Abbiamo poi implementato, ai fini di paragonare diversi approcci, un'altra soluzione per il routing, l'algoritmo genetico. Questo algoritmo, appartenente agli approcci meta-euristici, è molto usato in letteratura per risolvere problemi di VRP, principalmente per la sua capacità di non rimanere bloccato in una soluzione localmente - ma non globalmente - ottima. L'appendice alla fine di questo report descrive la sua implementazione.

[1] https://developers.google.com/optimization

Capitolo 6

Valutazione empirica

Trovandoci in un caso d'uso reale, possiamo valutare la bontà del sistema implementato in base alla funzione obiettivo da minimizzare, quella sulla distanza totale.

Implementare un sistema di valutazione richiede un nuovo processamento del dataset: dobbiamo simulare i turni di raccolta dei cestini, dunque dividere il dataset in finestre temporali.

In ogni finestra temporale di lunghezza w, dunque, ogni riga del dataset rappresenterà l'ultima rilevazione effettuata prima della fine della finestra per ogni cestino. Per decidere quali cestini svuotare adottiamo una semplice euristica, ovvero il Threshold Waste Level (TWL), una variabile che rappresenta il livello minimo di riempimento per cui il cestino deve essere svuotato. Tutti i cestini con livello di riempimento maggiore o uguale al TWL vengono dati in pasto prima al modello di clustering e poi all'algoritmo di routing. Per rispettare il vincolo che impone lo svuotamento di tutti i cestini almeno una volta al giorno, nell'ultima finestra di ogni giornata, oltre ai cestini pieni, vengono aggiunti alla lista anche tutti i cestini mai svuotati durante gli altri turni del giorno.

Abbiamo testato diverse configurazioni possibili per rispettare questo vincolo in modo più intelligente, in modo da minimizzare ancora di più le distanze percorse, ma siamo giunti alla conclusione che l'unico modo per soddisfarlo è implementare un modello di previsione del riempimento dei cestini; ciò tuttavia si è rivelato presto un compito fin troppo arduo, a causa principalmente della scarsa qualità del dataset fornito, che ci avrebbe costretti ad una fase di pulizia fin troppo onerosa. Abbiamo poi provato a distribuire i cestini non svuotati durante la giornata, simulando di avere un predittore del riempimento con accuratezza 100%: la differenza in termini di chilometraggio finale, abbiamo scoperto, è irrisoria rispetto all'aggiungere tali cestini all'ultimo turno, come facciamo nella soluzione finale implementata.

Sia il TWL che la grandezza delle finestre - in ore - sono parametri dati in input dall'utente. Il sistema dunque costruirà le finestre indicate e per ogni finestra eseguirà clustering dei cestini e routing dei veicoli. Ogni finestra darà quindi in output una distanza, calcolata per facilità di comprensione in km, corrispondente alla distanza percorsa dai veicoli in quel turno di raccolta. La distanza totale percorsa durante la simulazione, cioè per tutte le finestre, sarà il nostro KPI per valutare la performance della soluzione proposta.

Per dimostrare la bontà del sistema presentato, ovvero la combinazione di Shape-Controlled Clustering e algoritmo di Christofides - abbreviato *SC3* - confrontiamo

	Distanza media giornaliera (km)	**Distanza media per veicolo per turno (km)**
Baseline	640.76	27.11
Google Routing	622.06	26.32
SC2G	623.56	26.38
SC3	589.19	24.93

Tabella 6.1: Risultati dei test in termini di distanza media giornaliera e media per veicolo per turno.

questo approccio con i seguenti:

- Standard k-means e algoritmo di Christofides: la soluzione più semplice possibile, a cui ci riferiremo con il termine baseline;
- Google Routing: la libreria di Google permette di risolvere il problema anche senza clustering, suddividendo i nodi tra i veicoli con un approccio greedy;
- Shape-Controlled Clustering e algoritmo genetico per il routing (vedi appendice A) - abbreviato *SC2G*.

Il sistema è stato implementato interamente in Python, è completamente modulare e generalizzabile, ed è in grado di gestire sia soluzioni che prevedano il clustering che quelle senza clustering.

I parametri usati per la simulazione sono i seguenti: $w = 6$ ore, $C = 200$, $TWL = 4$, $V = 20$, periodo di test dal *01/09/2019* al *01/10/2019*.

I parametri usati per la funzione di distanza all'interno del clustering sono $\alpha = 1$ e $\beta = 0$.

I parametri usati per l'algoritmo genetico sono invece: $N = 500$, $e = 0.05$, $p = 0.1$, $G = 200$.

Il nodo considerato come deposito è quello indicato nelle specifiche, in posizione 45.5069182, 9.2684501.

6.1 Risultati

La tabella 6.1 presenta i risultati ottenuti. Come possiamo vedere, la soluzione implementata, *SC3*, migliora significativamente la distanza percorsa dai veicoli. I risultati ottenuti con l'algoritmo genetico sono potenzialmente migliorabili aumentando il numero di generazioni, come mostrato in figura A.1, ma c'è da considerare che già con questi parametri il routing è di diversi ordini di grandezza più lento rispetto a *SC3*.

La figura 6.1 invece confronta i percorsi generati a partire da un clustering "classico" con quelli generati a partire dal clustering di *SC3* o di *SC2G*: si evidenzia chiaramente come nel secondo caso i percorsi non siano sovrapposti, il veicolo cioè non "spreca" strada per raggiungere il suo cluster, ma raccoglie cestini nel tragitto.

Abbiamo quindi voluto testare la qualità della soluzione SC3 bilanciata, che permette di distribuire equamente i cestini tra i vari veicoli. Questa soluzione, ovviamente, performa meno bene (593.02 km - 25.09 km) del classico SC3, ma rimane comunque una soluzione migliore delle altre, pur rispettando un vincolo aggiuntivo.

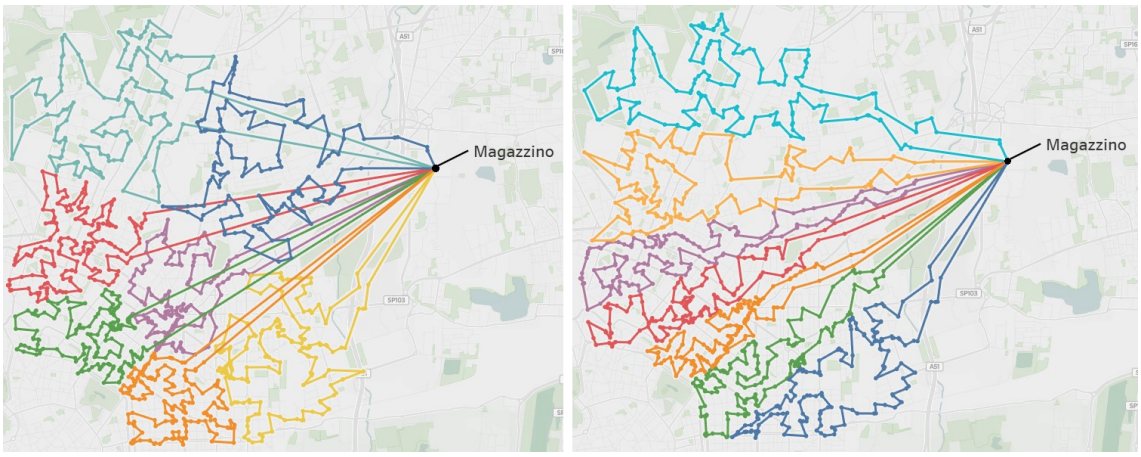

Figura 6.1: A sinistra i percorsi ottenuti a partire da un clustering classico. A destra i percorsi ottenuti con lo Shape-Controlled Clustering.

Per completezza decidiamo di eseguire un ulteriore test sulla scalabilità della soluzione proposta: in particolare vogliamo confrontare l'approccio con clustering migliore, *SC3*, con l'approccio migliore senza clustering, il Google Routing. Ci aspettiamo che l'approccio con clustering sia molto più efficiente in termini di scalabilità, proprio perchè suddivide un problema complesso come il VRP in multipli problemi più semplici. I risultati del test esposti nella figura 6.2 confermano quest'aspettativa.

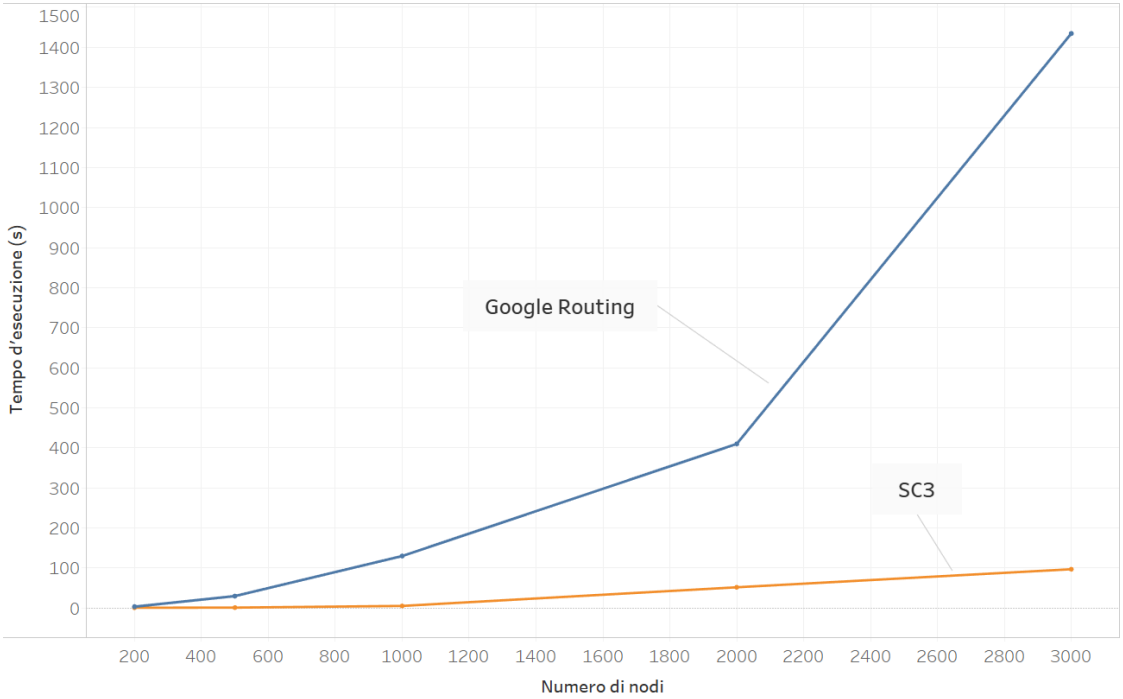

Figura 6.2: Andamento dei tempi d'esecuzione in base al numero di nodi dato in input agli algoritmi.

Capitolo 7

Conclusioni

Siamo molto soddisfatti dai risultati ottenuti, in particolar modo dalle performance che la nostra soluzione, *SC3*, ha generato. Siamo infatti partiti da una soluzione "facile", ma difficile da superare: quella di lasciar fare tutto a OR-Tools, la pluripremiata libreria di Google per la risoluzione di problemi complessi come il routing. Il nostro primo approccio, quello che usava un classico k-means, è stato fallimentare nel tentare di far meglio di Google, poichè tendeva a generare cluster globulari e percorsi sovrapposti. Tuttavia, grazie ad un attento studio della letteratura in merito al problema, siamo riusciti a proporre una soluzione innovativa, un ibrido delle tecniche che più ci avevano convinto durante il nostro studio; ci affidiamo comunque alla libreria di Google, ma ora solo per il routing: la nostra suddivisione dei cestini, infatti, ora è ben più efficiente rispetto a quella di Google, grazie principalmente alla nuova funzione di distanza implementata.

Siamo infatti in grado di creare dei cluster di forma allungata che, tra l'altro, tengano conto dei vincoli sulla capienza dei veicoli, e che generino poi percorsi non sovrapposti, come si vede in figura 6.1. Il tutto, poi, garantendo una scalabilità del sistema ottima, specie se confrontata alla soluzione interamente di Google, che dal grafico 6.2 sembra crescere esponenzialmente al'aumentare dei cestini.

Abbiamo poi proposto una soluzione alternativa, *SC2G*, che sfruttasse una delle "novità" in ambito di VRP, gli algoritmi metaeuristici: abbiamo infatti implementato una nostra versione dell'algoritmo genetico per il routing dei veicoli, che è sicuramente promettente e degna di essere considerata per eventuali sviluppi futuri. La nostra configurazione purtroppo non ci ha permesso di spremere a pieno la sua potenzialità di apprendimento, ma crediamo che con tanta potenza computazionale a disposizione potrebbe far meglio di *SC3* come risultati, sacrificando qualcosa in termini di tempi d'esecuzione.

Al di là di questo, crediamo fermamente che la nostra soluzione principale, *SC3*, sia una soluzione ben in grado di affrontare un problema reale di CVRP come quello di raccolta dei cestini. In un contesto ormai dominato da algoritmi meta-euristici complessi da progettare e che spesso richiedono per funzionare una base di codice non open-source, la nostra soluzione dimostra come un "vecchio" [1] approccio, quale è il "cluster first, route second", possa ancora dire la sua, ed anzi essere addirittura competitivo nei confronti degli approcci definiti come stato dell'arte.

[1] Il k-means, sebbene da noi pesantemente rivisitato, ha più di 50 anni; l'algoritmo di Christofides risale agli anni settanta.

Bibliografia

[1] Joaquın Bautista, Elena Fernández e Jordi Pereira. "Solving an urban waste collection problem using ants heuristics". In: *Computers & Operations Research* 35.9 (2008), pp. 3020–3033.

[2] Paul S Bradley, Kristin P Bennett e Ayhan Demiriz. "Constrained k-means clustering". In: *Microsoft Research, Redmond* 20.0 (2000), p. 0.

[3] Fahrettin Cakir, W Nick Street e Barrett W Thomas. "Revisiting Cluster First, Route Second for the Vehicle Routing Problem". In: *no. August* (2015).

[4] Nicos Christofides. *Worst-case analysis of a new heuristic for the travelling salesman problem.* Rapp. tecn. Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[5] Nicos Christofides, Aristide Mingozzi e Paolo Toth. "Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations". In: *Mathematical programming* 20.1 (1981), pp. 255–282.

[6] Geoff Clarke e John W Wright. "Scheduling of vehicles from a central depot to a number of delivery points". In: *Operations research* 12.4 (1964), pp. 568–581.

[7] SERAP ERCAN CÖMERT et al. "A new approach for solution of vehicle routing problem with hard time window: an application in a supermarket chain". In: *Sādhanā* 42.12 (2017), pp. 2067–2080.

[8] Georges A Croes. "A method for solving traveling-salesman problems". In: *Operations research* 6.6 (1958), pp. 791–812.

[9] Marco Dorigo, Mauro Birattari e Thomas Stutzle. "Ant colony optimization". In: *IEEE computational intelligence magazine* 1.4 (2006), pp. 28–39.

[10] Russell Eberhart e Joseph Sagaya Kennedy. "A new optimizer using particle swarm theory". In: *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science* (1995), pp. 39–43.

[11] Marshall L Fisher e Ramchandran Jaikumar. "A generalized assignment heuristic for vehicle routing". In: *Networks* 11.2 (1981), pp. 109–124.

[12] Michel Gendreau, Gilbert Laporte e René Séguin. "A tabu search heuristic for the vehicle routing problem with stochastic demands and customers". In: *Operations research* 44.3 (1996), pp. 469–477.

[13] Billy E Gillett e Leland R Miller. "A heuristic algorithm for the vehicle-dispatch problem". In: *Operations research* 22.2 (1974), pp. 340–349.

[14] Fred Glover. "Future paths for integer programming and links to ar tifi cial intelli g en ce". In: *Computers operations research* 13.5 (1986), pp. 533–549.

[15] Fred Glover e Manuel Laguna. "Tabu Search Background". In: *Tabu Search*. Springer, 1997, pp. 1–24.

[16] David E Goldberg e John Henry Holland. "Genetic algorithms and machine learning". In: (1988).

[17] MA Hannan et al. "Capacitated vehicle-routing problem model for scheduled solid waste collection and route optimization using PSO algorithm". In: *Waste management* 71 (2018), pp. 31–41.

[18] Anil K Jain. "Data clustering: 50 years beyond K-means". In: *Pattern recognition letters* 31.8 (2010), pp. 651–666.

[19] Gilbert Laporte, Yves Nobert e Serge Taillefer. "A branch-and-bound algorithm for the asymmetrical distance-constrained vehicle routing problem". In: *Mathematical Modelling* 9.12 (1987), pp. 857–868.

[20] Silvia Mazzeo e Irene Loiseau. "An ant colony algorithm for the capacitated vehicle routing". In: *Electronic Notes in Discrete Mathematics* 18 (2004), pp. 181–186.

[21] "MiniZinc Challenge 2019 Results". In: (2019). URL: `https://www.minizinc.org/challenge2019/results2019.html`.

[22] Denis Naddef e Giovanni Rinaldi. "Branch-and-cut algorithms for the capacitated VRP". In: *The vehicle routing problem*. SIAM, 2002, pp. 53–84.

[23] D Otoo, SK Amponsah, C Sebil et al. "Capacitated clustering and collection of solid waste in kwadaso estate, Kumasi". In: *Journal of Asian Scientific Research* 4.8 (2014), pp. 460–472.

[24] David Pisinger e Stefan Ropke. "A general heuristic for vehicle routing problems". In: *Computers & operations research* 34.8 (2007), pp. 2403–2435.

[25] Ingo von Poser e Adel R Awad. "Optimal routing for solid waste collection in cities by using real genetic algorithm". In: *2006 2nd International Conference on Information & Communication Technologies*. Vol. 1. IEEE. 2006, pp. 221–226.

[26] Martin Reed, Aliki Yiannakou e Roxanne Evering. "An ant colony algorithm for the multi-compartment vehicle routing problem". In: *Applied Soft Computing* 15 (2014), pp. 169–176.

[27] Xueyuan Wang e Hongyu Zhu. "Two-stage Algorithm for Capacitated Vehicle Routing Problem." In: *Journal of Engineering Science & Technology Review* 11.2 (2018).

Appendice A

Algoritmo genetico

Sebbene per il routing esistano metodi statistici, quale l'algoritmo Christofides [4], negli ultimi anni si è osservato che algoritmi che si ispirano alla biosfera [16] [10] [9] riescono a convergere verso risultati subottimali in tempi abbastanza brevi e con una buona accuratezza. Fra questi, l'algoritmo genetico è uno dei più usati in letteratura per la risoluzione del problema della raccolta dei rifiuti.

L'algoritmo genetico si ispira al processo di evoluzione genetica, secondo il quale una popolazione si evolve col passare delle generazioni in funzione della minimizzazione dello sforzo fatto per raggiungere un determinato scopo, tipico della quotidianità di quella popolazione. In particolare, date una popolazione di N individui, ciascuno composto da g geni, un numero di generazioni G, una percentuale di elitismo e, una probabilità di mutazione p e una funzione f da minimizzare, le fasi del processo di evoluzione su cui si basa l'algoritmo sono le seguenti:

1. Si scelgono gli individui da cui generare la successiva generazione. In particolare, una *elite* comprendente il migliore $(e \times N)\%$ degli individui (quelli che meglio minimizzano f), viene selezionata a priori. Il restante $((1-e) \times N)\%$ viene scelto casualmente dall'intera popolazione;

2. Degli individui selezionati, l'*elite* viene automaticamente selezionata per la successiva generazione, mentre i restanti individui vengono generati tramite *crossover* fra quelli selezionati in precedenza. Il crossover è il processo mediante il quale, dati due individui e un numero casuale c, $1 < c < g$, viene generato un individuo avente i primi c geni del primo genitore e i restanti $g-c$ geni del secondo. In sostanza, si tratta del processo di riproduzione;

3. La nuova popolazione ha probabilità p di subire una mutazione, ossia lo scambio di due geni casuali. Dato un numero casuale m, se $m < p$ l'individuo viene mutato;

4. Si ripetono i passi 1-3 per il numero di generazioni G fissato in partenza.

Per applicare l'algoritmo al problema in esame è sufficiente interpretare i cestini come nodi di un grafo e scegliere come numero di geni il numero di cestini in ogni cluster. Gli altri parametri sono scelti a discrezione dell'utente. Inoltre, bisogna fornire in input all'algoritmo, oltre alla lista delle posizioni dei cestini, la posizione inziale/finale dalla quale i veicoli partono e alla quale fanno ritorno dopo aver collezionato i rifiuti di tutti i cestini nel cluster. Dunque, è sufficiente modificare l'algoritmo in modo tale da:

- Etichettare i nodi del grafo con numeri interi da 1 a g e il nodo iniziale/finale con 0: dunque, ciascun individuo consiste in una sequenza di numeri interi rappresentanti etichette di nodi;
- Aggiungere a ogni individuo i geni relativi ai nodi iniziale e finale;
- scegliere f quale somma delle distanze (non ha importanza il tipo di formula usata per calcolarla) fra ciascuna coppia di geni/nodi in un individuo;
- Impedire al processo di mutazione di scambiare geni relatvi ai nodi iniziale e finale.

Dopo G generazioni, il percorso la cui lunghezza risulta minima fra gli individui dell'ultima generazione, viene selezionato come percorso minimo.

Il percorso finale generato viene ottimizzato attraverso le euristiche migliorative intra-route 2-opt [8] e 3-opt.

Il grafico A.1 mostra l'andamento della distanza di un singolo percorso di un veicolo generato con l'algoritmo genetico: come possiamo vedere, per un numero di generazioni elevato l'algoritmo genetico converge verso un minimo.

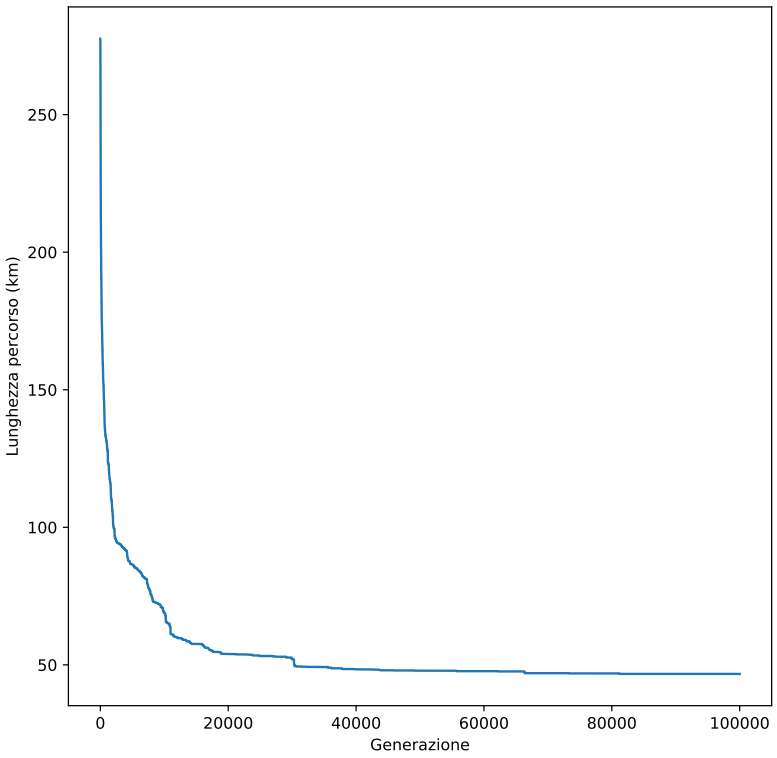

Figura A.1: Andamento della distanza prodotta dall'algoritmo genetico in base al numero di generazioni.

Poichè l'algoritmo genetico, a causa dei molti loop presenti nel codice, risultava inizialmente molto lento e non scalabile (oltre 300 secondi per un singolo percorso), si è deciso di sfruttare la libreria Python per il calcolo parallelo *numba*. Dopo aver reso

eseguibili in parallelo le funzioni più onerose dell'algoritmo (breeding, mutazione, calcolo della lunghezza, passo di ottimizzazione 2-opt) i risultati sono migliorati di un fattore pari a circa 10: si è passati da oltre 300 secondi (senza ottimizzazione 2-opt/3-opt) a poco più di 20 (con ottimizzazione), per $G = 200$, $N = 500$ $p = 0.1$, $e = 0.05$, $g = 150$).