Politecnico di Torino

Dipartimento di Ingegneria Gestionale e della Produzione

Corso di Laurea Triennale in Ingegneria Gestionale - Classe L8

A.A. 2025/2026

Applicazione per l'ottimizzazione degli itinerari di campagna elettorale

Relatore Giuseppe Bruno Averta

Candidato Edoardo Russu S312556

Sommario

1. Proposta di progetto

1.1 Titolo del progetto

Applicazione per l'ottimizzazione degli itinerari di campagna elettorale.

1.2 Descrizione del problema affrontato

In campagna elettorale il tempo è limitato a causa di spostamenti, incontri ed imprevisti. Nell'ambito di una provincia, con decine di comuni, visitarli tutti non è possibile, perciò, si rende necessaria la selezione delle tappe da fare e in che ordine, in base alla collocazione territoriale, alla popolazione ed alla cultura dei cittadini. Ogni comune ha un diverso potenziale, dal punto di vista elettorale (abitanti, aventi diritto di voto e storico di affluenza alle urne).

È fondamentale, quindi, trovare un insieme di comuni ed inserirli in un itinerario, che massimizzi il "potenziale", senza superare il tempo a disposizione, che è limitato. L'applicazione proposta affronta questo problema di selezione e ordinamento sul grafo e propone un piano realistico.

1.3 Descrizione della rilevanza gestionale del problema

Dal punto di vista gestionale, l'interesse è ottimizzare le risorse messe a disposizione del candidato (tempo, personale e budget), per ottenere la massima produttività, dal punto di vista elettorale. L' applicazione proposta funge da strumento di supporto decisionale, tramite criteri (pesi e vincoli), mostra un'analisi approfondita del territorio (elettori e storico di affluenza) e confronta più piani alternativi. Inoltre, migliora la programmazione operativa e gestisce le azioni organizzative, al fine di raggiungere gli obiettivi preposti, riducendo sprechi (tempo e budget) e aumentando al massimo il confronto con gli elettori.

1.4 Descrizione preliminare dei data-set per la valutazione

Per lo sviluppo dell'elaborato sono stati utilizzati tre principali insiemi di dati, scelti per la loro affidabilità e coerenza, rispetto agli obiettivi del progetto.

Il primo è l'Elenco dei Comuni italiani, fornito dall'ISTAT. Questo data-set costituisce la base anagrafica dell'intero lavoro, in quanto contiene per ciascun comune il codice ISTAT, la denominazione ufficiale, la provincia e la regione di appartenenza. Tali informazioni sono state impiegate sia come riferimento identificativo univoco, sia come chiave di join per l'integrazione con le altre tabelle. La versione utilizzata è aggiornata al 1° gennaio 2025.

Il secondo data-set è quello relativo all'Affluenza alle Elezioni Politiche del 2022, pubblicato dal portale ufficiale Eligendo. Per ogni comune, il data-set riporta il numero di elettori, di votanti e la percentuale di affluenza finale. Queste informazioni sono state utilizzate come variabili per stimare il potenziale di partecipazione associato a ciascun nodo del grafo, contribuendo così al calcolo del punteggio.

Infine, è stato impiegato il dataset delle Coordinate geografiche correnti, anch'esso proveniente da fonti ISTAT. Esso fornisce le latitudini e longitudini dei comuni, consentendo di determinare la posizione spaziale dei nodi e di calcolare le distanze e i tempi di percorrenza, in funzione della velocità media calcolata.

1.5 Descrizione preliminare degli algoritmi coinvolti

L'elaborato si sviluppa in tre fasi principali: assegnazione del punteggio ai comuni, costruzione del grafo e calcolo del percorso ottimale.

Nella prima fase, ogni comune appartenente ad una stessa provincia viene valutato in base a due indicatori: numero di elettori e percentuale di affluenza. Entrambi i valori vengono normalizzati su una scala da 0 a 1; l'affluenza viene inoltre invertita, in modo da attribuire punteggi più alti ai comuni con minore partecipazione, considerando più alto il potenziale elettorale. La combinazione dei due fattori avviene tramite una media pesata (70% elettori e 30% affluenza), da cui deriva un punteggio finale espresso su scala 1–10.

Successivamente, viene costruito un grafo non orientato, pesato e completamente connesso, in cui i nodi rappresentano i comuni e gli archi indicano i tempi di percorrenza stimati. Le distanze tra i comuni vengono calcolate attraverso la formula di Haversine, che utilizza le coordinate geografiche (latitudine e longitudine), per ottenere una misura precisa della distanza in chilometri.

Nell'ultima fase viene applicato un algoritmo ricorsivo con backtracking, per individuare il percorso con potenziale massimo, ossia la combinazione di tappe che massimizza il punteggio complessivo nel rispetto dei vincoli di tempo e numero di comuni visitabili.

2. Descrizione dettagliata del problema affrontato

I social media hanno, radicalmente, trasformato il concetto di campagna elettorale, modificando il modo in cui i politici ed i partiti comunicano ed interagiscono con gli elettori.

Tuttavia, la campagna elettorale basata sui comizi, sui contatti diretti con gli stakeholders e l'elettore in genere, continua a rappresentare un supporto fondamentale all'attività propagandistica.

Il contatto diretto stabilisce un livello di fiducia personale molto elevato. L'elettore percepisce l'impegno, l'ascolto e la concretezza del candidato, che va oltre la 'teca' curata dei profili online. Il politico deve trasformare gli incontri in occasioni di dialogo e far sentire la propria vicinanza rispetto alle istanze specifiche di quel territorio, in modo che il cittadino si senta protagonista attivo delle scelte che si andranno ad operare.

Ciò si rende necessario soprattutto in virtù della disaffezione che si è registrata ultimamente nei confronti della politica e delle istituzioni, che ha determinato un significativo calo dell'affluenza alle urne. Alle elezioni politiche del 1976 hanno votato il 93,4% degli aventi diritto; a partire dal 1979 si è registrata una progressiva riduzione della partecipazione al voto, che ha raggiunto il record negativo nel 2022 con il 63,8 %.

Le riunioni e le cene elettorali sono cruciali per mobilitare la base dei sostenitori. Il porta a porta e gli eventi locali raggiungono quei cittadini che non usano i social o che vengono esclusi dai meccanismi del targeting tradizionale e del micro-targeting. L'utilizzo dei social benché in crescita non è universale.

Secondo i dati ISTAT 2024 la percentuale degli over 75 anni, che utilizzano Internet è del 31%; per la fascia di età che va dai 65 ai 74 anni è del 68%.

L'astensionismo è un fenomeno preoccupante che mette in serio pericolo le democrazie occidentali e di cui bisogna contrastare le cause.

Per invertire la tendenza, sarà necessario un profondo ripensamento del rapporto tra politica e cittadini, con un ritorno alla trasparenza e alla partecipazione attiva, ma sarà necessaria anche l'adozione di misure concrete che possano attenuare un fenomeno che, lentamente, sta uccidendo le nostre democrazie liberali, limitando notevolmente la portata del principio affermato nel primo articolo della nostra Carta Costituzionale, che sancisce che la sovranità appartiene al popolo, che la esercita nelle forme e nei limiti della Costituzione.

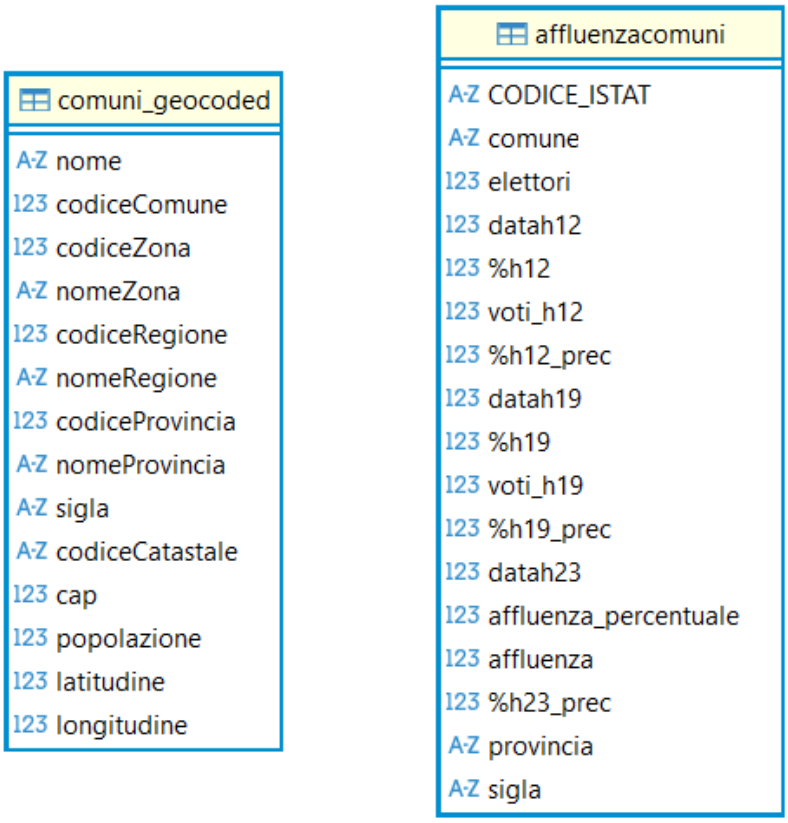
L'estensione territoriale di alcuni collegi, unitamente ai tempi ristretti della campagna elettorale, costituisce delle criticità, per cui si rende necessaria l'ottimizzazione dei tempi, così come la razionalizzazione dei costi.

L'obiettivo del mio programma tende, appunto, a massimizzare l'azione politica, ampliando i contatti 'fisici' con l'elettore. La razionalizzazione degli spostamenti mira ad ottenere il massimo risultato con il minimo dispendio di tempo e denaro. A partire dal 2013 il finanziamento pubblico ai partiti è stato gradualmente abolito e, nel contempo, sono stati introdotti dei limiti di spesa, per cui la riduzione degli esborsi elettorali è diventata un elemento imprescindibile.

Il percorso generato dal programma tiene conto delle distanze e quindi dei tempi di percorrenza tra i comuni, degli aventi diritto al voto e dell'affluenza relativa alle elezioni nazionali del 2022.

3. Descrizione del data-set utilizzato

Il data-set utilizzato è composto da due tabelle:



La tabella **comuni_geocoded** è composta da:

- nome → riporta il nome del comune;
- codiceComune → riporta il codice ISTAT del comune, verrà utilizzato come chiave di join per le due tabelle;
- codiceZona → riporta un valore numerico da 1 a 5, rappresenta il codice della macro-zona geografica. Es: Isole (Sardegna e Sicilia) codice 5;
- nomeZona → riporta il nome della macro-zona geografica. Es: Piemonte fa parte della macro-zona "Nord-ovest";
- codiceRegione → riporta un valore numero da 1 a 20 che segue la numerazione ISTAT;
- nomeRegione→ riporta il nome della regione;
- codiceProvincia→ riporta un valore numerico da 1 a 110 assegnato in ordine storico (non alfabetico). Es: Piemonte codice 1;
- nomeProvincia→ riporta il nome della provincia;
- sigla→ riporta la sigla della provincia;
- codiceCatastale→ riporta il codice catastale, un codice a 4 cifre alfanumeriche asseganto a ogni comune italiano dall' Agenzia delle Entrate. Serve per riconoscere i comuni nei sistemi catastali e fiscali;
- cap→ riporta il Codice di Avviamento Postale è un codice composto da 5 cifre, usato come riferimento per il recapito della posta;

- popolazione→ riporta il numero di abitanti aggiornato al 1° gennaio 2025;
- latitudine→ riporta la latitudine espressa in gradi decimali;
- longitudine→ riporta la longitudine espressa in gradi decimali.

La tabella **affluenzacomuni** è composta da:

- CODICE_ISTAT→ riporta il codice ISTAT, un identificativo numerico assegnato a ogni comune italiano dall'ISTAT (Istituto Nazionale di Statistica). Serve per riconoscere in modo univoco ogni comune;
- comune→ riporta il nome del comune;
- elettori→ riporta il numero degli aventi diritto di voto aggiornato al 1° gennaio 2022;
- datah12→ riporta data e ora della rilevazione dei voti, relativa alle elezioni nazionali del 2022 ore 12;
- %h12→ riporta la percentuale delle votazioni rispetto al numero di elettori alle ore 12;
- voti_h12→ riporta il numero delle votazioni alle ore 12;
- %h12_prec→ riporta la percentuale delle votazioni rispetto al numero di elettori alle ore 12, relativa alla precedente elezione, per fornire un'ulteriore visione dell'andamento dell'elezione attuale;
- datah19→ riporta data e ora della rilevazione dei voti, relativa alle elezioni nazionali del 2022 ore 19;
- %h19→ riporta la percentuale delle votazioni rispetto al numero di elettori alle ore 19;
- voti_h12→ riporta il numero delle votazioni alle ore 12;
- %h19_prec→ riporta la percentuale delle votazioni rispetto al numero di elettori alle ore 19, relativa alla precedente elezione;
- datah23→ riporta data e ora della rilevazione dei voti, relativa alle elezioni nazionali del 2022 ore 23;
- affluenza_percentuale→ → riporta la percentuale delle votazioni rispetto al numero di elettori alle ore 23. Questo valore verrà utilizzato nel calcolo del punteggio;
- affluenza→ riporta il numero delle votazioni alle ore 12;
- %h23_prec→ → riporta la percentuale delle votazioni rispetto al numero di elettori alle ore 23, relativa alla precedente elezione;
- provincia→ riporta il nome della provincia;
- sigla→ riporta la sigla della provincia;

Per l'utilizzo dei dati sono stati cambiati i seguenti nomi:

- %h23 → affluenza_percentuale;
- voti_h23 → affluenza;

e sono stati aggiunti due valori nella seconda tabella, tramite un join con chiave il codice ISTAT del comune:

- provincia;
- sigla.

4. Strutture dati e algoritmi utilizzati

Il programma da me elaborato utilizza la struttura MVC (Model, View, Controller). Questo approccio permette di organizzare il codice in modo ordinato, facilitando eventuali modifiche.

La struttura è composta da:

- Il **Model** contiene la parte logica e gestionale del programma. Si occupa dell'elaborazione dei dati e fornisce i metodi relativi alla gestione dei dati provenienti dal database.
- La **View** rappresenta l'interfaccia utente, gestisce il rapporto utente-programma generando la pagina con cui l'utente comunica col programma.
- Il **Controller** ha la funzione di intermediario tra Model e View. Dalla View riceve gli input dell'utente e richiama i metodi del Model per elaborarli. Successivamente aggiorna la View in modo che l'utente possa vedere i risultati delle elaborazioni effettuate.

A queste componenti si aggiunge il **DAO** (Data Access Object), che serve per gestire l'accesso ai dati contenuti nel database scelto in modo sicuro, aprendo una connessione che richiede l'inserimento del nome utente e della password dell'ambiente di gestione del database utilizzato (in questo caso DBeaver).

Contiene inoltre i metodi utili per effettuare operazioni sui dati. Questi calcoli vengono elaborati dal programma all'interno dell'ambiente di sviluppo (PyCharm), in modo da ridurre la complessità delle operazioni effettuate nel Model.

4.1 Teoria dei Grafi

Il programma utilizza la teoria dei grafi, una disciplina, ramo della matematica e dell'informatica, che studia le relazioni tra determinati oggetti tramite la struttura del "Grafo".

Un grafo è definito attraverso una coppia (V, E), dove V è un insieme finito di vertici ed E è un insieme finito di archi.

Gli archi possono essere:

- **Orientati,** nel caso in cui la relazione tra i due nodi sia unidirezionale (A → B);
- **Non orientati,** se la relazione è bidirezionale (A → B, B → A).

Il grafo può essere:

- **Connesso,** se tra ogni coppia di nodi esiste almeno un percorso disponibile;
- **Completamente connesso,** se ogni nodo è collegato direttamente a tutti gli altri nodi presenti nel grafo;
- **Disconnesso,** se esiste un nodo o un insieme di nodi che non sono collegati ad altri nodi, per cui partendo da un nodo non esiste un percorso disponibile per raggiungere tutti i nodi del grafo.

```python
def buildGraph(self, provincia):
    self._grafo.clear()
    self._provincia = provincia
    self._nodes = DAO.getAllComuni(provincia)
    self.calcolaPunteggio(self._nodes)
    self._grafo.add_nodes_from(self._nodes)
    for n in self._nodes:
        self._idMap[int(n.codice)] = n
    self._edges = DAO.getAllEdges(provincia)
    for edge in self._edges:
        self._grafo.add_edge(
            self._idMap[int(edge.codice1)],
            self._idMap[int(edge.codice2)],
            weight=edge.distanza
        )
    return self._grafo
```

Il metodo mostrato in figura si trova all' interno del Model, alla pressione del bottone "Analizza provincia" il programma genera un grafo completamente connesso. I nodi sono rappresentati dai comuni facenti parte della provincia selezionata, mentre il peso degli archi corrisponde alla distanza tra i comuni calcolata con la seguente formula di Haversine:

```
2 * 6371.0088 * ASIN(
SQRT(
POW(SIN(RADIANS(c2.latitudine - c1.latitudine) / 2), 2) +
COS(RADIANS(c1.latitudine)) *
COS(RADIANS(c2.latitudine)) *
POW(SIN(RADIANS(c2.longitudine - c1.longitudine) / 2), 2))) AS distanza
```

I nodi sono rappresentati dalla classe Comune:

```
@dataclass
class Comune :
    codice :int
    nome : str
    provincia : str
    codiceP : str
    elettori :int
    affluenza : int
    affluenza_percentuale :float
    punteggio :int
```

Alla creazione del grafo viene assegnato per ogni comune un punteggio, inizialmente posto a 0 tramite il seguente metodo:

```
def calcolaPunteggio(self, comuni):
    self._minElettori = min(c.elettori for c in comuni)
    self._maxElettori = max(c.elettori for c in comuni)
    self._minAffluenza = min(c.affluenza for c in comuni)
    self._maxAffluenza = max(c.affluenza for c in comuni)

    for comune in comuni:
        self._normaElettori = ((comune.elettori - self._minElettori) /
                               (self._maxElettori - self._minElettori))
        self._normaAffluenza = 1 - ((comune.affluenza - self._minAffluenza) /
                                    (self._maxAffluenza - self._minAffluenza))
        media = (0.7 * self._normaElettori) + (0.3 * self._normaAffluenza)
        comune.punteggio = round(media * 10, 2)
    return
```

Inizialmente viene normalizzato il punteggio relativo a elettori e affluenza all' interno della stessa provincia, successivamente viene calcolata una media pesata (70% elettori, 30% 1-affluenza), infine il punteggio viene rappresentato da 1 a 10.

Inoltre questo metodo aggiorna l'idMap precedentemente creato, assegnando alla chiave univoca (codice ISTAT) il nodo corrispondente, in modo da facilitare la successiva ricerca dei nodi nell' algoritmo ricorsivo.

4.2 Algoritmo ricorsivo con backtracking

Un algoritmo ricorsivo è costituito da un insieme di procedimenti che utilizzano il concetto di ricorsione, ossia la capacità di un algoritmo a richiamare sè stesso.

L'algoritmo è formato da:

- Condizione finale, la condizione per cui l'algoritmo deve fermarsi;
- Processo ricorsivo, l'algoritmo richiama sé stesso aggiornando i dati in input del metodo;
- Backtracking, l'azione con cui l'algoritmo riesce ad analizzare tutte le soluzioni disponibili.

All'interno del mio programma è stato elaborato questo algoritmo ricorsivo:

```
def ricorsione(self, parziale, giorniUtilizzati, oreUtilizzate, giorniMax, tappe):
    if len(parziale) == tappe:
        if self.calcolaPeso(parziale) > self._punteggioMax:
            self._punteggioMax = self.calcolaPeso(parziale)
            self._percorsoOttimo = copy.deepcopy(parziale)
            self._tempoMin = self.calcolaTempo(parziale)
        if self.calcolaPeso(parziale) == self._punteggioMax and self.calcolaTempo(parziale) < self._tempoMin:
            self._punteggioMax = self.calcolaPeso(parziale)
            self._percorsoOttimo = copy.deepcopy(parziale)
            self._tempoMin = self.calcolaTempo(parziale)
        return
    ultimo = parziale[-1]
    for nodo in self._grafo.nodes:
        if nodo in parziale:
            continue
        distanza = float(self._grafo[ultimo][nodo].get("weight",0.0))
        tempoViaggio = distanza / (self.getVelocitaMedia(distanza))
        tot = tempoViaggio + self._tempoSosta
        if oreUtilizzate + tot <= self._oreGiornaliere:  # stesso giorno
            parziale.append(nodo)
            self.ricorsione(parziale, giorniUtilizzati, oreUtilizzate + tot, giorniMax, tappe)
            parziale.pop()
        else:
            if giorniUtilizzati + 1 <= giorniMax and tot <= self._oreGiornaliere: # cambio giorno
                parziale.append(nodo)
                self.ricorsione(parziale, giorniUtilizzati + 1, tot, giorniMax, tappe)
                parziale.pop()
    return
```

Il metodo richiede in input:

- **parziale** → lista di comuni già visitati;
- **giorniUtilizzati** → giorni impiegati per visitare tutti i comuni all' interno della lista;
- **oreUtilizzate** → ore impiegate;
- **giorniMax** → giorni a disposizione dell'utente;
- **tappe** → numero di comuni che l'utente vuole visitare.

La condizione finale è rappresentata dal raggiungimento del numero di comuni che l'utente si è prefissato di visitare.

A questo punto se il punteggio ottenuto risulta maggiore del punteggio massimo fin ora raggiunto dall'algoritmo, il metodo aggiorna il punteggio, salvando la lista dei comuni e aggiornando il tempo

utilizzato. Nel caso il punteggio ottenuto sia uguale al punteggio massimo fin ora raggiunto, ma il tempo impiegato sia minore del tempo minimo salvato in precedenza, il metodo aggiorna la lista dei comuni e il tempo minimo.

Successivamente, tramite un ciclo for, il metodo esegue l'algoritmo ricorsivo di ogni nodo non presente in parziale, calcolando distanza e tempo necessario.

Il tempo di percorrenza è calcolato in base alla distanza e alla velocità media, calcolata a seconda della distanza:

```
def getVelocitaMedia(self,distanza):
    # calcolo la velocità media in base alla distanza tra i comuni
    if distanza < 5:
        velocita = 40
    elif distanza < 20:
        velocita = 60
    elif distanza < 80:
        velocita = 80
    else:
        velocita = 100
    return velocita
```

In quanto all' aumentare della distanza, corrisponde un aumento della velocità media.

Infine, il metodo richiama sé stesso per poi fare backtracking. Se il tempo necessario allo spostamento e alla sosta (fissata di 2h) non supera il limite di ore giornaliero (fissato di 8h) il metodo si limita a richiamarsi, in caso contrario controlla se ha raggiunto il limite dei giorni fissati dall'utente e alla variabile giorniUtilizzati aggiunge 1.

5. Diagramma delle classi principali

Classe Model

Variabili:

- provincia
- grafo
- idMap
- province
- tempoSosta
- oreGiornaliere
- percorsoOttimo
- punteggioMax
- tempoMin

Metodi:

- buildGraph(provincia)
- calcolaPunteggio(comuni)
- trovaPercorso(partenza, giorniMax, nTappe)
- ricorsione(parziale, giorniUtilizzati, oreUtilizzate, giorniMax, tappe)
- calcolaPeso(parziale)
- calcolaTempo(parziale)
- getOrdinati()
- getVelocitaMedia(distanza)

Classe Controller

Variabili:

- view
- model

Metodi:

- handleAnalizzaProvincia()
- fullProvince()
- handleCalcolaPercorso()

Classe DAO

Metodi:

- getAllComuni(provincia)
- getAllEdges(provincia)
- getProvince()

Classe View

Variabili:

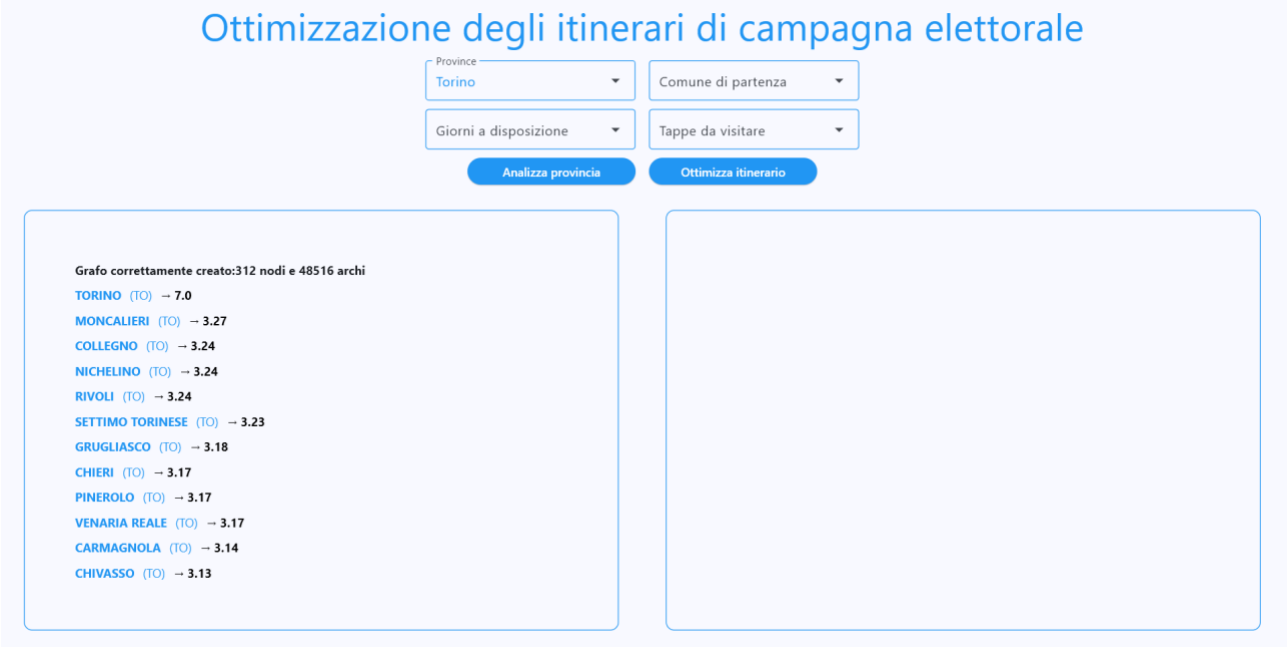
- page
- controller
- title
- btnAnalizzaProvincia
- btnCalcolaPercorso
- province
- partenza
- NGiorni
- NTappe
- lv1
- lv2

Metodi:

- load_interface()
- set_controller(controller)
- update_page()

6. Videate dell'applicazione

Interfaccia iniziale



Alla pressione del bottone "Analizza provincia"

Alla pressione del bottone “Ottimizza itinerario”

Errore riportato in caso l’utente premesse il bottone “Analizza provincia” senza aver prima selezionato una provincia

Errore riportato in caso non esista un percorso disponibile che rispetti i vincoli dell'utente

Per maggiori approfondimenti segue il link del video esplicativo: https://youtu.be/KZw8UhXrbFw

8. Risultati sperimentali

Il mio programma genera un grafo completamente connesso, per cui la complessità cresce quadraticamente col numero dei nodi e si esprime con $O(n^2)$.

Infatti, per massimizzare le prestazioni del calcolatore è necessario scegliere vincoli (giorni e tappe) di piccole dimensioni, in modo da arrivare il prima possibile alla condizione finale dell'algoritmo ricorsivo.

Seguono tempistiche relative per l'elaborazione del percorso a seconda della grandezza della provincia.

Grafo con ~ 10 nodi:

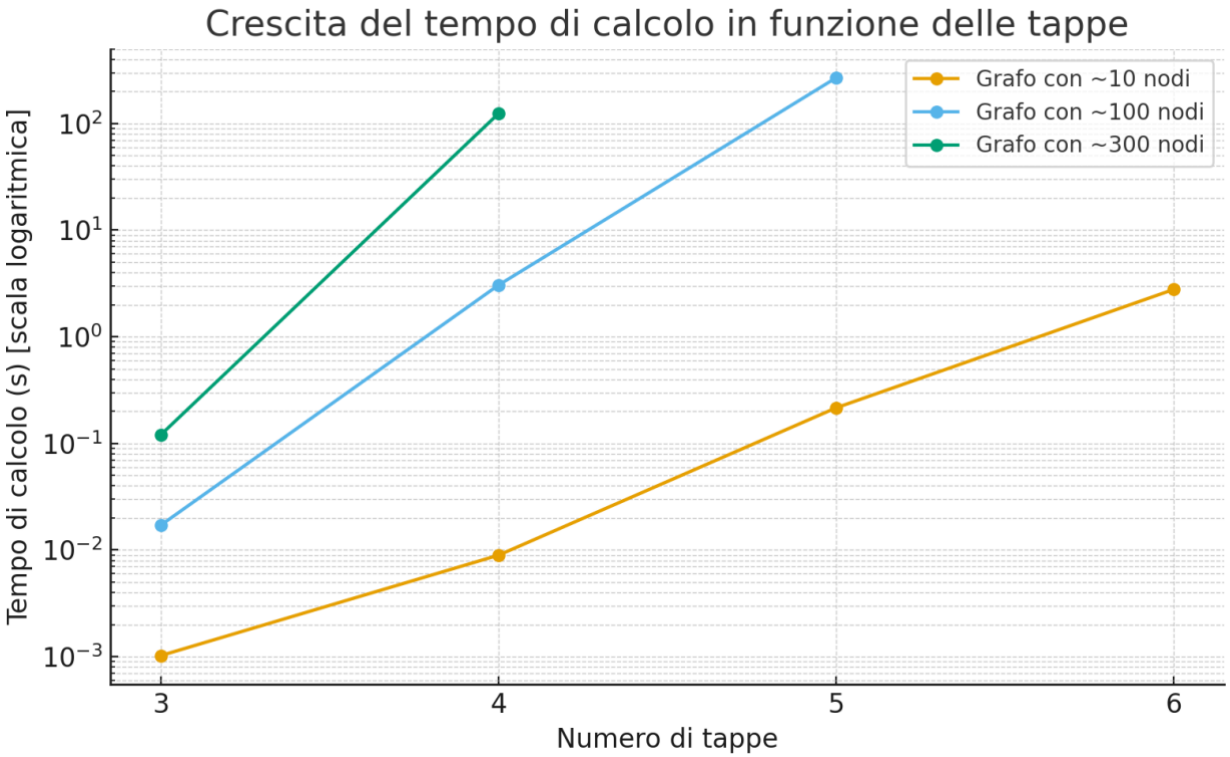
Giorni	Tappe	Tempo(s)
1	3	0.0010254383087158203
2	4	0.008946895599365234
2	5	0.21554303169250488
3	6	2.7755870819091797

Grafo con ~ 100 nodi:

Giorni	Tappe	Tempo(s)
1	3	0.017163991928100586
2	4	3.05318021774292
2	5	268.09962368011475

Grafo con ~ 300 nodi:

Giorni	Tappe	Tempo(s)
1	3	0.12096071243286133
2	4	124.13312554359436



Questo grafico ci mostra come cresce il tempo in base al numero di tappe, e ci fa notare subito come per il grafo con 300 nodi l'algoritmo esploda già a raggiungimento di 4 tappe.

9. Valutazioni e conclusioni

Con il mio lavoro ritengo di poter fornire all'azienda (partito e/o coalizione) uno strumento utile a migliorare il risultato elettorale, rendendo più efficace e concreta l'azione politica e propagandistica, in un'ottica di economicità.

L'applicazione da me programmata si basa su alcuni elementi oggettivi forniti dall'azienda, quali il Comune di partenza, il tempo a disposizione ed il numero di tappe da raggiungere.

Le criticità sono rappresentate dalla complessità di calcolo, proporzionale al numero di comuni all'interno della Provincia selezionata, che richiede un calcolatore dalle alte prestazioni.

10. Licenza

Il lettore è libero di:

1. Condividere — riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare questo materiale con qualsiasi mezzo e formato
2. Modificare — remixare, trasformare il materiale e basarsi su di esso per le sue opere
3. Il licenziante non può revocare questi diritti fintanto che lui rispetti i termini della licenza.

Alle seguenti condizioni:

1. Attribuzione — Deve riconoscere una menzione di paternità adeguata , fornire un link alla licenza e indicare se sono state effettuate delle modifiche . Può fare ciò in qualsiasi maniera ragionevole possibile, ma non con modalità tali da suggerire che il licenziante avalli lui o il suo utilizzo del materiale.
2. Non Commerciale — Non può utilizzare il materiale per scopi commerciali .
3. Stessa Licenza — Se remixa, trasforma il materiale o si basi su di esso, deve distribuire i suoi contributi con la stessa licenza del materiale originario.
4. Divieto di restrizioni aggiuntive — Non può applicare termini legali o misure tecnologiche che impongano ad altri soggetti dei vincoli giuridici su quanto la licenza consente loro di fare.

Note:

Non è tenuto a rispettare i termini della licenza per quelle componenti del materiale che siano in pubblico dominio o nei casi in cui il suo utilizzo sia consentito da una eccezione o limitazione prevista dalla legge.

Non sono fornite garanzie. La licenza può non conferirgli tutte le autorizzazioni necessarie per l'utilizzo che si prefigge. Ad esempio, diritti di terzi come i diritti all'immagine, alla riservatezza e i diritti morali potrebbero restringere gli usi che si prefigge sul materiale.