Tesina Ottimizzazione Combinatoria

Vito Ruggiero e Aldo Rambaudo

1. Shortest Path Problem

In un mondo in cui il tempo è spesso il nostro bene più prezioso, l'ottimizzazione delle rotte e la pianificazione dei percorsi diventano sempre più cruciali.
Immaginiamo di dover pianificare un viaggio per raggiungere un'importante riunione di lavoro o di dover consegnare beni essenziali in un'area congestionata; in entrambi i casi, trovare il percorso più veloce ed efficiente diventa un obiettivo fondamentale.
È qui che entra in gioco il Time-Dependent Shortest Path (TDSP), un concetto che rivoluziona l'approccio alla pianificazione dei percorsi.
Il classico Shortest Path (SP) ha svolto un ruolo fondamentale nell'ottimizzazione dei percorsi per decenni. Tuttavia, questo modello presenta delle limitazioni significative quando si tratta di affrontare situazioni del mondo reale in cui il tempo gioca un ruolo cruciale.
Il modello SP suppone che i costi per attraversare gli archi di un grafo siano fissi, ignorando completamente le fluttuazioni temporali che possono verificarsi. Immaginiamo di essere bloccati nel traffico o di viaggiare su una rete di trasporto pubblico con orari variabili. In queste situazioni, il tempo di percorrenza di un arco può variare notevolmente durante la giornata, rendendo il classico SP inadatto per fornire soluzioni ottimali.

Questo è solo un esempio delle sfide che il mondo reale ci presenta, dove il tempo è una variabile cruciale e il classico SP fallisce nell'affrontare tali complessità.

Il Time-Dependent Shortest Path (TDSP) è stato sviluppato per affrontare le limitazioni del SP classico, considerando il tempo come una variabile dinamica.
Questo ci permette di calcolare i percorsi ottimali che tengono conto dei cambiamenti che possono verificarsi nei costi degli archi, come ad esempio il traffico stradale o gli orari di punta dei mezzi pubblici.
A seguire, esploreremo in dettaglio la struttura del TDSP, le sue applicazioni e gli algoritmi utilizzati per risolvere questo problema complesso.

2. Differenze tra Shortest Path e Time-Dependent Shortest Path

Per comprendere a pieno il concetto di Time-Dependent Shortest Path (TDSP), è essenziale confrontarlo con il classico problema dello Shortest Path (SP).
In questa parte, esploreremo le principali differenze tra questi due approcci, mettendo in evidenza come il TDSP si distingua per l'utilizzo del tempo come variabile chiave.

2.1 Costi Variabili nel Tempo

Nel SP classico, i costi associati agli archi del grafo sono considerati fissi; questo significa che il tempo necessario per attraversare un arco rimane costante indipendentemente dal momento in cui si effettua il percorso.
Questa semplificazione può portare a soluzioni non realistiche in quanto, nella realtà, i costi variano nel tempo, per esempio a causa del traffico stradale o in base agli orari dei mezzi pubblici.
Il TDSP invece riconosce le variazioni temporali dei costi.
Ogni arco del grafo temporale è associato a una funzione di costo temporale che rappresenta il tempo di percorrenza in funzione del momento della giornata.
Ciò consente di calcolare percorsi ottimali che tengono conto delle fluttuazioni dei costi nel tempo, garantendo soluzioni più realistiche e pratiche.

2.2 Orario di Partenza

Nel SP classico, non è necessario specificare un orario di partenza, il calcolo del percorso ottimale si basa esclusivamente sulla struttura statica del grafo, senza considerare il momento in cui si effettua il percorso.
Al contrario, nel TDSP, è fondamentale specificare l'orario di partenza desiderato, perché esso determina il punto di partenza per il calcolo del percorso ottimale, tenendo conto delle variazioni dei costi nel tempo a partire da quell'orario.
Quindi, ricapitolando, il TDSP fornisce soluzioni che sono ottimali rispetto ai costi variabili nel tempo e questo significa che le soluzioni calcolate dal TDSP sono più aderenti alla realtà, poiché considerano le variazioni dei costi durante il percorso.

2.3 Applicazioni Pratiche

Il SP classico è spesso utilizzato in scenari in cui i costi degli archi sono statici, come la navigazione su mappe digitali con informazioni fisse sul traffico.
Il TDSP trova applicazioni in situazioni reali in cui il tempo è un fattore critico, come la pianificazione di percorsi ottimali in città congestionate, la gestione della logistica con orari variabili dei trasporti e la risoluzione di emergenze che richiedono il calcolo tempestivo di percorsi dinamici.

3. Struttura del Time-Dependent Shortest Path (TDSP)

Per comprendere meglio il Time-Dependent Shortest Path (TDSP), è essenziale esaminarne la struttura alla base.
In questa parte, verrà illustrato il concetto di TDSP, come viene modellato e le sue basi teoriche.

3.1 Modello Concettuale del TDSP

Il TDSP è un'estensione del classico problema dello Shortest Path (SP) che tiene conto delle variazioni dei costi nel tempo.
La sua struttura si basa su due componenti principali:

Grafo Temporale: a differenza del grafo utilizzato nel SP classico, il TDSP utilizza un grafo temporale. In questo grafo, ogni arco è associato a una funzione di costo temporale che rappresenta il costo (tempo) necessario per attraversare quell'arco in un determinato momento.
Queste funzioni di costo temporale possono variare durante la giornata in base a fattori come il traffico stradale o gli orari dei mezzi pubblici.
Un Grafo Temporale può essere definito come un grafo pesato G = (V, E), dove V rappresenta l'insieme dei nodi ed E l'insieme degli archi.
Ogni arco (u, v) in E è associato a una funzione di costo temporale $c(u, v, t)$, dove u e v sono nodi collegati da quell'arco e t rappresenta il tempo.

Orario di Partenza: nel TDSP, è necessario specificare l'orario di partenza desiderato.
Questo determina il momento da cui inizierà il calcolo del percorso ottimale.
Il TDSP cerca, quindi, un percorso che minimizzi il tempo totale di percorrenza, tenendo conto delle variazioni dei costi in funzione del tempo.

3.2 Rappresentazione del Grafo

La rappresentazione del grafo nel Time-Dependent Shortest Path è essenziale per tenere traccia delle variazioni temporali dei costi.
Ciò può essere fatto in diverse maniere:

- **Archi Temporali Ponderati:** ogni arco del grafo è associato a un peso temporale che rappresenta il costo (tempo) per attraversare quell'arco in un momento specifico.
  Questi pesi possono essere pre-calcolati o aggiornati in tempo reale a seconda delle variazioni dei costi.
- **Intervallo di Validità:** ogni arco può avere un intervallo di validità temporale durante il quale il suo costo è applicabile. Ad esempio, un arco potrebbe avere un costo diverso durante le ore di punta rispetto alle ore in cui il traffico è minore.
- **Grafi Dinamici:** in alcuni casi, è possibile utilizzare grafi dinamici che si adattano automaticamente alle variazioni temporali dei costi. Questi grafi possono essere più complessi da gestire, ma consentono una modellazione più precisa delle variazioni.

3.3 Funzione di Costo Temporale

La funzione di costo temporale è al centro del TDSP ed è responsabile della rappresentazione dei costi variabili nel tempo.
Questa funzione associa un costo temporale (solitamente il tempo di percorrenza) a ciascun arco del grafo in base al momento della giornata.
Sia G = (V, E) il grafo pesato, dove V rappresenta l'insieme dei nodi ed E rappresenta l'insieme degli archi.
Sia, poi, $c(u, v, t)$ la funzione di costo temporale associata all'arco (u, v) nel grafo, dove:
- u rappresenta il nodo di partenza dell'arco;
- v rappresenta il nodo di destinazione dell'arco;
- t rappresenta il tempo, che è una variabile continua.

La funzione di costo temporale $c(u, v, t)$ è una funzione matematica che associa il costo necessario per attraversare l'arco (u, v) al tempo t. Questa funzione rappresenta la variazione dei costi nel tempo.
Le forme specifiche della *funzione di costo temporale* possono variare a seconda delle condizioni del mondo reale e delle specifiche esigenze dell'applicazione.
Tuttavia, generalmente, si tratta di funzioni continue che possono essere lineari, non lineari o complesse. Più nel dettaglio, la funzione di costo temporale può essere:

- **Lineare:**
  $c(u, v, t) = a(u, v) * t + b(u, v)$; dove $a(u, v)$ rappresenta la pendenza della funzione lineare e b(u, v) rappresenta il termine di intercetta.
  Un approccio comune consiste nell'utilizzare funzioni di costo lineari, dove il costo varia in modo proporzionale al tempo. Questo è utile per rappresentare aumenti costanti nel tempo, come il traffico che aumenta gradualmente durante le ore di punta.
- **Non Lineare:**
  $c(u, v, t) = f(u, v, t)$; dove $f(u, v, t)$ è una funzione non lineare che modella variazioni complesse dei costi nel tempo.
  In alcuni casi, le variazioni dei costi potrebbero non essere uniformi nel tempo, ad esempio a causa di congestioni stradali improvvise. Per modellare questo tipo di variazioni più complesse, possono essere utilizzate le funzioni di costo non lineari, in cui il costo per attraversare un arco varia in modo non lineare rispetto al tempo. Tali funzioni possono assumere diverse forme, ad esempio logaritmica, esponenziale, polinomiale, ecc.
- **A Scaglioni:**
  $c(u, v, t) = c\_i(u, v)$ per t in $[t\_i, t\_\{i+1\})$; dove $c\_i(u, v)$ rappresenta il costo dell'arco (u, v) durante l'intervallo di tempo $[t\_i, t\_\{i+1\})$. Questa funzione è utile per modellare variazioni di costo discrete in diverse fasce orarie.
  Le funzioni di costo a scaglioni suddividono il tempo in intervalli discreti e assegnano un costo fisso a ciascun intervallo. I costi rimangono costanti all'interno di ogni intervallo e possono cambiare bruscamente quando si passa da un intervallo all'altro, come il traffico nei pressi di diversi punti di interesse di una città che cambia in base alla fascia oraria.

La scelta della forma specifica della funzione di costo temporale dipende dalla natura del problema e dalla precisione richiesta nella modellazione delle variazioni dei costi nel tempo.

4.Algoritmi per la risoluzione del TDSP

La risoluzione del Time-Dependent Shortest Path (TDSP) è una sfida complessa a causa della variabilità dei costi nel tempo; tuttavia, esistono diversi algoritmi che sono stati sviluppati per calcolare soluzioni efficienti per il TDSP.
In questa parte, esamineremo alcuni di questi algoritmi e le loro principali caratteristiche. *Dijkstra e A** (in particolare *Dijkstra con Heap Modificato* e *A* Time-Dependent*) sono stati utilizzati nell'applicativo da noi sviluppato, che verrà successivamente illustrato; *Contraction Hierarchies* (sia *Standard* che *Time-Dependent*), invece, verrà solo menzionato in linea teorica.

Prima di esaminare gli algoritmi però, potrebbe essere utile definire il problema relativo al Time-Dipendent Shortest Path.

Il problema del Time-Dependent Shortest Path (TDSP) può essere definito come un grafo pesato G = (V, E), dove V rappresenta l'insieme dei nodi ed E l'insieme degli archi.
Ogni arco (*u, v*) in E è associato a una funzione di costo temporale *c*(*u, v, t*), dove *u* e *v* sono nodi collegati da quell'arco, e t rappresenta il tempo.
Nel contesto del TDSP, consideriamo un nodo di partenza *s* e un nodo di destinazione *d* all'interno del grafo G. Inoltre, è necessario specificare un istante di tempo di partenza *t_s* da cui inizia la ricerca del percorso ottimale.

La soluzione al problema è rappresentata dal percorso ottimale P = (*v*_1, *v*_2, ..., *v_k*) che collega il nodo di partenza *s* al nodo di destinazione *d* e minimizza il tempo totale di percorrenza.
Il tempo totale T(P) di percorrenza lungo il percorso P è dato dalla somma dei costi temporali lungo ogni arco visitato: T(P) = Σ[c(v_i, v_{i+1}, t_i)], dove v_i rappresenta il nodo i-esimo nel percorso P e t_i rappresenta il tempo in cui l'arco (v_i, v_{i+1}) viene attraversato.

4.1 Dijkstra

4.1.1 Dijkstra Standard
L'algoritmo Dijkstra è utilizzato per trovare il percorso più breve da un nodo di origine a tutti gli altri nodi in un grafo pesato. Esso memorizza un insieme di nodi con i percorsi più brevi conosciuti e utilizza una coda di priorità (spesso implementata come una coda min-heap) per selezionare il nodo che permette di ottenere il percorso più breve in modo efficiente. L'algoritmo procede iterativamente fino a quando non si ottiene il percorso più breve per ogni nodo.

4.1.1 Dijkstra con Heap Modificato
Dijkstra con Heap Binario Modificato (Time-Dependent) è una variante di Dijkstra che tiene conto della variazione dei pesi degli archi nel tempo. Questa variazione può rappresentare situazioni reali come il traffico stradale che cambia nel corso della giornata.

Le principali differenze tra l'algoritmo di Dijkstra modificato con heap binario per il TDSP e l'algoritmo di Dijkstra standard sono:

- i costi degli archi, che possono variare nel tempo, come nel caso del traffico stradale che cambia a diverse ore del giorno. In Dijkstra Standard, invece, i costi degli archi restano costanti;
- la coda di priorità, che in questo caso si basa su una struttura dati come un heap binario per gestire i nodi da esplorare in base al tempo variabile di percorrenza, mentre in Dijkstra Standard di basa esclusivamente sulla loro distanza. Questo rende l'algoritmo più efficiente rispetto all'implementazione ingenua di Dijkstra Standard;
- il calcolo dei costi temporali totali per raggiungere ciascun nodo, che in questo caso avviene considerando la variazione temporale degli archi.

In sintesi, l'algoritmo di Dijkstra modificato con heap binario è adattato per gestire problemi di TDSP, dove i costi degli archi sono dipendenti dal tempo. Questa variazione è essenziale per trovare il percorso più breve tenendo conto delle variazioni temporali nei costi, come il traffico stradale o altri fattori che influenzano il tempo di percorrenza.

Questo algoritmo può essere suddiviso in diverse fasi:

Inizializzazione: l'algoritmo parte da un nodo di origine e inizializza la distanza da questo nodo a tutti gli altri nodi con un valore infinito, tranne la distanza dal nodo di origine a se stesso, che è zero. Queste distanze rappresentano la lunghezza dei percorsi più brevi conosciuti fino a quel momento. Nella versione con Heap Binario Modificato, ogni arco nel grafo è associato a una funzione temporale che indica come cambiano i pesi degli archi nel tempo. Ad esempio, questa funzione potrebbe rappresentare il tempo di percorrenza di una strada in diverse ore del giorno.

Heap Binario Modificato: invece di utilizzare una coda di priorità standard, Dijkstra con Heap Binario Modificato utilizza una struttura dati speciale, chiamata "heap binario modificato" o "temporal heap". Questa struttura dati tiene traccia dei nodi in base a due criteri: la distanza dalla sorgente e il tempo attuale. Quindi, ogni nodo nell'heap ha una coppia (distanza, tempo) associata.

Tempo-Dipendente: durante l'esplorazione del grafo, l'algoritmo tiene traccia del tempo attuale e aggiorna dinamicamente i pesi degli archi in base alla funzione temporale associata a ciascun arco. Questo significa che i pesi degli archi possono cambiare mentre l'algoritmo sta esplorando il grafo, riflettendo le variazioni reali nel tempo (come il traffico stradale che aumenta o diminuisce).

Ricalcolo dei Percorsi: A causa delle variazioni dei pesi degli archi nel tempo, l'algoritmo potrebbe dover ricalcolare i percorsi più brevi in modo dinamico. Questo comporta l'esplorazione di nuovi nodi e archi per trovare il percorso più breve al tempo attuale.

*4.2 Algoritmo A**

4.2.1 A Standard*

L'algoritmo A* (pronunciato "A star") è un algoritmo di ricerca euristica utilizzato per trovare il percorso più breve tra due punti in un grafo ponderato. Questo algoritmo è ampiamente utilizzato in problemi di percorso, come la pianificazione del percorso in un gioco, la navigazione GPS o la robotica.
L'obiettivo principale dell'algoritmo A* è trovare il percorso ottimale tra un nodo di partenza e un nodo di destinazione in un grafo ponderato, minimizzando la somma dei costi degli archi attraversati.

4.2.2 A Time-Dependent*

L'algoritmo A* Time-Dependent (TD-A*) è una variante dell'algoritmo A* utilizzata per risolvere problemi di Time-Dependent Shortest Path (TDSP), in cui i costi degli archi variano nel tempo. Esso si differenzia dall'A* Standard per:

- i costi degli archi, che possono variare nel tempo, mentre nell'A* standard sono costanti;
- le stime euristiche, che sono calcolate in modo dinamico durante l'esecuzione, tenendo conto delle variazioni temporali nei costi. Nell'A* standard, invece, esse sono costanti o calcolate in base a una mappa statica;
- i nodi con i valori di distanza minimi, che devono essere aggiornati in base alle variazioni temporali dei costi. Questo aggiornamento non è, invece, necessario nell'A* standard.

Quindi, l'obiettivo principale del TD-A* è trovare il percorso più breve tra due nodi in un grafo ponderato, tenendo conto delle variazioni temporali nei costi degli archi.

L'algoritmo A* Time-Dependent è suddiviso in diverse fasi chiave:

Inizializzazione: in questa fase, vengono inizializzate le distanze da un nodo iniziale a tutti gli altri nodi a "infinito", ad eccezione del nodo iniziale stesso, il cui valore di distanza è impostato a 0.

***Esecuzione Principale*:** l'algoritmo esegue un ciclo principale finché ci sono nodi da esplorare. Viene estratto il nodo con il valore di distanza minimo dalla coda e diventa il nodo attuale. Se il nodo attuale è uguale al nodo finale, l'algoritmo è completato e viene costruito il percorso ottimo; altrimenti, se il nodo attuale non è uguale al nodo finale, l'algoritmo continua a cercare il percorso ottimo. In questa situazione, l'algoritmo esegue una serie di passaggi.

- **Estrazione del Nodo Minimo**: l'algoritmo estrae il nodo con il valore di distanza minimo (distanza percorsa finora + stima del costo rimanente) dalla coda, questo nodo diventa il nodo attuale.
- **Esplorazione dei Nodi Vicini:** l'algoritmo esamina tutti i nodi collegati al nodo attuale tramite archi (nodi vicini).

- **Aggiornamento delle Distanze e dei Punteggi F:** per ogni nodo vicino, l'algoritmo calcola un punteggio F considerando la distanza percorsa fino a quel momento e una stima euristica del costo rimanente per raggiungere il nodo finale.
  Se il nuovo punteggio F è migliore (più basso) di quello precedentemente calcolato per il nodo vicino, le distanze e il punteggio F vengono aggiornati.
- **Inserimento nella Coda con Priorità:** dopo l'aggiornamento delle distanze e dei punteggi F, il nodo vicino viene inserito nella coda con priorità.
- **Ripetizione:** l'algoritmo continua a iterare su questi passaggi fino a quando il nodo attuale diventa uguale al nodo finale (se esiste un percorso valido) o fino a quando esaurisce tutte le possibili opzioni senza trovare un percorso valido.

Costruzione del Percorso Ottimo: Se il nodo finale è raggiunto, l'algoritmo ricostruisce il percorso ottimo risalendo dai nodi finali ai nodi precedenti utilizzando le informazioni sui predecessori, memorizzate durante l'esecuzione.

In sintesi, l'obiettivo principale dell'algoritmo A* è trovare il percorso più breve (o il percorso con il costo minimo) da un nodo iniziale a un nodo finale in un grafo pesato. Questo viene fatto considerando sia il costo reale (distanza percorsa) sia una stima euristica (distanza euclidea) del costo rimanente. Questa stima euristica è ciò che distingue A* dagli altri algoritmi di ricerca come Dijkstra.

4.3 Contraction Hierarchies (CH)

4.3.1 Contraction Hierarchies (CH) Standard

Contraction Hierarchies (CH) Standard è un algoritmo di routing utilizzato per trovare i percorsi più brevi in reti di grafi statici. Questo algoritmo è particolarmente efficace nelle reti stradali o in altre reti di trasporto, ma può essere applicato a qualsiasi rete di grafi statica. La sua principale caratteristica distintiva è la creazione di una gerarchia dei nodi nella rete per accelerare il calcolo dei percorsi più brevi.

La prima fase dell'algoritmo coinvolge la creazione di una gerarchia dei nodi nella rete. Questa gerarchia è basata sull'importanza dei nodi per il calcolo dei percorsi più brevi. In sostanza, i nodi vengono raggruppati in base a determinati criteri.

Durante la creazione della gerarchia, i nodi vengono contratti in gruppi di diverse dimensioni. Questo significa che alcuni nodi diventano nodi "superiori" che implicano gruppi di nodi "inferiori". Questa contrazione dei nodi consente di creare una gerarchia di nodi che rappresenta la rete in modo più efficiente.

Una volta creata la gerarchia, l'algoritmo utilizza questa struttura per calcolare i percorsi più brevi. Essa permette di escludere rapidamente alcune opzioni di percorso che non porterebbero al percorso ottimale.

Una volta trovato il percorso più breve, l'algoritmo restituisce il percorso completo, indicando gli archi e i nodi da attraversare.

La velocità e l'efficienza di CH Standard derivano dalla creazione della sopracitata gerarchia dei nodi, la quale permette di evitare il calcolo di percorsi inutili e di ridurre così significativamente il tempo necessario per trovare il percorso ottimale.

Esiste una versione time-dependent (TD-CH) di questo algoritmo che estende il CH per gestire i percorsi più brevi in reti dove il tempo di percorrenza tra i nodi cambia in base all'orario o ad altri fattori temporali.

4.3.2 Contraction Hierarchies Time-Dependent (CHTD)

Contraction Hierarchies Time-Dependent (CHTD) è la variante dell'algoritmo CH standard che consente di calcolare percorsi più brevi in reti di grafi dove i tempi di percorrenza sugli archi variano nel tempo, ad esempio, in reti stradali con dati sul traffico che cambiano in base all'orario o altri fattori temporali. Questo algoritmo è progettato per affrontare dati temporali dinamici e fornire percorsi ottimali che tengano conto delle variazioni temporali.

Il processo inizia con la creazione della gerarchia dei nodi nella rete. Come per CH Standard, la gerarchia viene creata in base all'importanza dei nodi per il calcolo dei percorsi più brevi. Tuttavia, a differenza di CH standard, CH Time-Dependent tiene conto dei dati temporali durante la creazione della gerarchia.

Nella fase di pre-elaborazione, vengono integrati i dati temporali nella rete. Ogni arco del grafo è associato a informazioni temporali che indicano il tempo di percorrenza in diverse ore del giorno o in altre fasce temporali.

Questi dati temporali sono fondamentali per calcolare percorsi ottimali che considerino le variazioni temporali.

Quando un utente richiede un percorso ottimale tra due punti sulla rete, specifica anche l'orario di partenza desiderato.
L'algoritmo utilizza la gerarchia e i dati temporali per determinare il percorso più breve in base all'orario specificato. Questo significa che il percorso potrebbe essere diverso a seconda dell'orario scelto, tenendo conto delle condizioni del traffico o di altri fattori temporali.
Una volta calcolato il percorso ottimale, l'algoritmo restituisce il percorso completo, indicando gli archi e i nodi da attraversare, insieme al tempo previsto per ogni segmento del percorso in base all'orario specificato.

L’algoritmo CH Time-Dependent è particolarmente utile per applicazioni in tempo reale come i servizi di navigazione GPS che tengono conto del traffico stradale in diverse ore del giorno. Questo algoritmo offre la flessibilità di adattare i percorsi in base alle condizioni temporali, garantendo che gli utenti possano ottenere percorsi ottimali in qualsiasi momento. La chiave per il suo funzionamento è la combinazione della gerarchia con i dati temporali integrati nella rete per calcolare percorsi più brevi Time-Dependent in modo efficiente.

5. Applicazioni Pratiche al problema del Time-Dependent Shortest Path

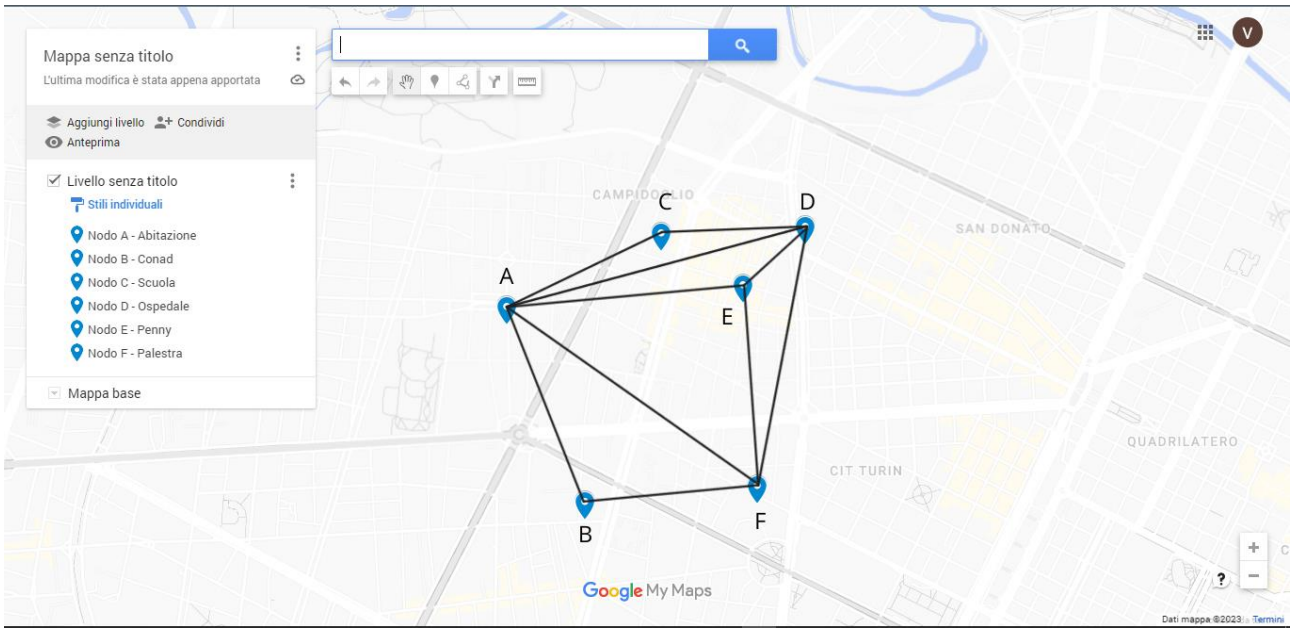
Nell'ambito della risoluzione dei problemi dei percorsi ottimali in reti dinamiche, la combinazione degli algoritmi A*, Dijkstra Modificato e Contraction Hierarchies Time-Dependent riveste un ruolo di fondamentale importanza.
Questa sinergia di algoritmi rappresenta un notevole avanzamento nel campo della navigazione intelligente e dell'ottimizzazione dei percorsi nelle reti di grafi che variano nel tempo.
Le reti dinamiche, come le reti stradali o di trasporto pubblico, presentano variazioni temporali nei tempi di percorrenza degli archi, dovute a fattori come il traffico stradale, le condizioni meteorologiche o la disponibilità di risorse. La ricerca di percorsi ottimali in queste reti richiede un approccio sofisticato che tenga conto di queste fluttuazioni.
Gli algoritmi A* e Dijkstra sono ampiamente noti e utilizzati per calcolare i percorsi più brevi in reti statiche. Tuttavia, per affrontare reti dinamiche, è necessario adattarli per considerare i dati temporali in modo efficace. Questa relazione esplorerà l'applicazione di tali algoritmi in Python per risolvere il problema del Time-Dependent Shortest Path in reti dinamiche.
In particolare, ci concentreremo su come questi algoritmi vengono adattati per gestire le variazioni temporali, consentendo agli utenti di calcolare percorsi ottimali in base all'orario di partenza o ad altri fattori temporali. Esamineremo, poi, le modifiche apportate agli algoritmi base, l'integrazione dei dati temporali e la loro implementazione pratica in Python.
Questa relazione fornirà una panoramica delle applicazioni degli algoritmi A* e Dijkstra Modificato per la risoluzione di problemi di percorso ottimale, dimostrando l'importanza di tali approcci nell'ottimizzazione della navigazione e nell'adattamento ai cambiamenti temporali nelle reti di grafi.

5.1 Rappresentazione del Grafo

Per testare gli algoritmi di percorso con variazioni temporali, abbiamo intrapreso un approccio basato su dati reali, utilizzando la città di Torino (precisamente zona Campidoglio/Parella), come scenario di studio. Abbiamo creato un grafo di base utilizzando nodi che rappresentano punti di interesse reali nella città. Ogni punto d'interesse è stato selezionato con cura e, per ciascuno di essi, abbiamo acquisito dati di *distanza* e *tempistiche di percorrenza* utilizzando strumenti come Google Maps e altri servizi di navigazione.
A seguire, i sei nodi principali che abbiamo considerato nel nostro studio, correlati dalla loro corrispondente ubicazione:

- Nodo A (Casa): Via Domodossola, 35
- Nodo B (Conad City): Via Bardonecchia, 5/C
- Nodo C (Scuola Pubblica): Via Balme, 42
- Nodo D (Ospedale Maria Vittoria): Via Luigi Cibrario, 72
- Nodo E (Penny Market): Piazza Risorgimento, 20
- Nodo F (Palestra Freestyle SSD): Via Enrico Cialdini, 13

Per ciascun nodo, abbiamo raccolto dati accurati sulle distanze fisiche tra di essi e le relative tempistiche di percorrenza in diverse condizioni di traffico.



Di seguito viene riportata una tabella riassuntiva con le relative distanze e tempistiche calcolate con Google Maps.
Tali tempistiche, ovviamente, non sono legate a una determinata fascia oraria con le relative condizioni di traffico, per cui il percorso e il tempo impiegato per raggiungere un nodo sono sempre gli stessi in qualsiasi momento della giornata.

KM/time	Nodo A	Nodo B	Nodo C	Nodo D	Nodo E	Nodo F
Nodo A	/	900m/13min	1km/13min	1,5km/22min	1km/10min	1,3km/18min
Nodo B	900m/13min	/	Non diretto	Non diretto	Non diretto	900m/12min
Nodo C	1km/13min	Non diretto	/	600m/9min	Non diretto	Non diretto
Nodo D	1,5km/22min	Non diretto	600m/9min	/	600m/9min	1,2km/12min
Nodo E	1km/10min	Non diretto	Non diretto	600m/9min	/	900m/13min
Nodo F	1,3km/18min	Non diretto	900m/12min	1,2km/12min	900m/13min	/

Tali misurazioni ci hanno permesso di essere il più possibile fedeli alla realtà e, partendo da questo grafo, abbiamo successivamente ipotizzato come queste condizioni potessero evolversi in diverse circostanze.

Gli aspetti da prendere in considerazione potrebbero essere:

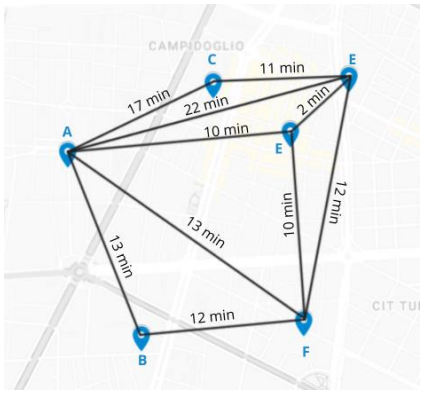
- *condizioni meteorologiche* come pioggia, neve o nebbia che potrebbero influenzare la velocità di percorrenza delle strade;
- *eventi speciali* come manifestazioni, chiusure stradali per lavori in corso o eventi sportivi che potrebbero alterare temporaneamente le rotte disponibili, creando deviazioni e ritardi.
- *traffico stradale,* il quale potrebbe cambiare nei pressi di determinati punti di interesse della città in base alla fascia oraria.

Tra gli aspetti sopracitati, abbiamo scelto il *traffico stradale*, in quanto sembrava il più facile da ipotizzare e, inoltre, è anche uno dei problemi più facili da riscontrare quotidianamente.

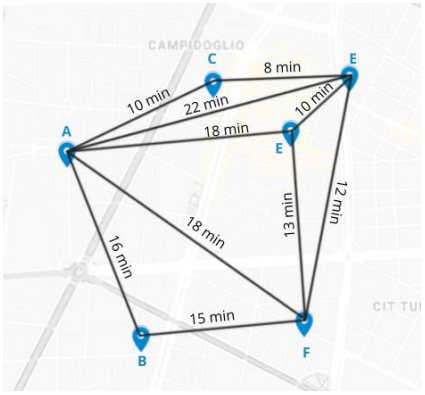
Durante le ore di punta, le strade possono diventare congestionate, rallentando notevolmente i tempi di percorrenza rispetto al resto della giornata; pertanto, abbiamo ipotizzato che a seconda delle fasce orarie, alcune zone potessero essere più o meno popolate, con un maggiore o minore traffico stradale e, perciò, alcuni percorsi risulterebbero migliori rispetto ad altri.

Successivamente, abbiamo creato tre diverse versioni del grafo basate su fasce orarie specifiche: 08:00, 12:00 e 18:00.
Tali versioni riflettono le variazioni temporali delle condizioni stradali in diversi orari della giornata. Di conseguenza, le tempistiche di percorrenza tra i nodi sono state modificate per rispecchiare i cambiamenti nella congestione del traffico o in altre influenze temporali.

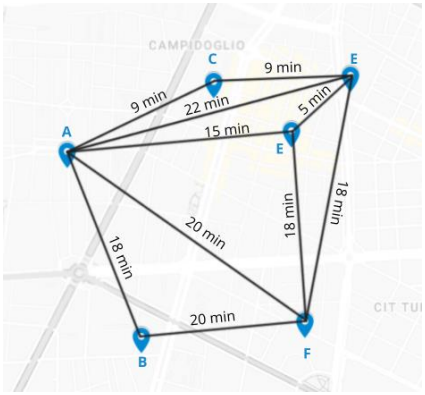
Grafo ore 08:00



Grafo ore 12:00



Grafo ore 18:00



Queste tre versioni del grafo ci hanno permesso di testare gli algoritmi A* e Dijkstra Modificato in scenari realistici, tenendo conto delle variazioni temporali che si verificano durante la giornata.

L'obiettivo principale del nostro studio consiste nel valutare l'efficacia degli algoritmi nella ricerca dei percorsi ottimali in condizioni temporali dinamiche e nell'adattamento alle diverse fasce orarie, contribuendo così a una migliore comprensione delle loro prestazioni nella gestione di reti dinamiche del mondo reale.

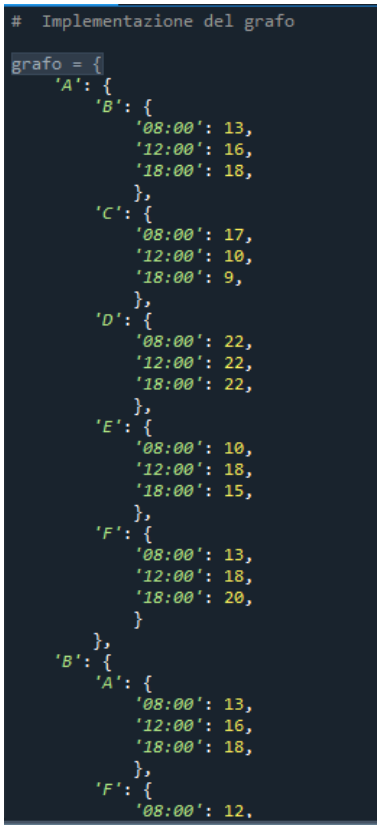
5.2 Applicativo Python

L'applicativo Python da noi sviluppato rappresenta un passo significativo nella gestione intelligente dei percorsi nelle reti TDSP. La capacità di calcolare percorsi ottimali in base alle variazioni temporali offre numerosi vantaggi per la mobilità e la logistica. Questo progetto dimostra il potenziale delle soluzioni software nell'ottimizzazione della navigazione in situazioni reali, tenendo conto delle sfide temporali dinamiche che caratterizzano le reti di trasporto moderne.

1. **Creazione della Struttura del Grafo TDSP:**

Abbiamo iniziato con la creazione di una struttura dati che rappresenti in modo accurato il grafo TDSP. Questa struttura tiene conto di variabili chiave come nodi, archi e dati temporali associati agli archi. Ogni arco è associato a informazioni temporali che riflettono le variazioni nei tempi di percorrenza in diverse fasce orarie.

L'immagine a seguire mostra l'implementazione del grafo, nella quale vengono memorizzate le tempistiche di percorrenza di ogni nodo con i nodi ad esso adiacenti nelle diverse fasce orarie.



```
#  Implementazione del grafo

grafo = {
    'A': {
        'B': {
            '08:00': 13,
            '12:00': 16,
            '18:00': 18,
            },
        'C': {
            '08:00': 17,
            '12:00': 10,
            '18:00': 9,
            },
        'D': {
            '08:00': 22,
            '12:00': 22,
            '18:00': 22,
            },
        'E': {
            '08:00': 10,
            '12:00': 18,
            '18:00': 15,
            },
        'F': {
            '08:00': 13,
            '12:00': 18,
            '18:00': 20,
            }
        },
    'B': {
        'A': {
            '08:00': 13,
            '12:00': 16,
            '18:00': 18,
            },
        'F': {
            '08:00': 12,
```

2. **Interazione con l'Utente:**

Abbiamo implementato una funzione interattiva che consente all'utente di specificare tre elementi fondamentali:

- Il luogo (nodo) di partenza.

- Il luogo (nodo) di destinazione.
- La fascia oraria in cui desidera pianificare il percorso.

```
# Interazione con l'utente

nodo_iniziale = input("Inserisci il nodo di partenza: ").strip().upper()
nodo_finale = input("Inserisci il nodo di destinazione: ").strip().upper()
orario_desiderato = input("Seleziona l'orario (8:00, 12:00 o 18:00): ").strip()
tempo_percorrenza, percorso = dijkstra_temporale(grafo, nodo_iniziale, nodo_finale, orario_desiderato)

if tempo_percorrenza is not None:
    print(f"Il tempo di percorrenza da {nodo_iniziale} a {nodo_finale} alle {orario_desiderato} è di {tempo_percorrenza} minuti.")
    print(f"Percorso: {' -> '.join(percorso)}")
else:
    print(f"Non è stato trovato un percorso da {nodo_iniziale} a {nodo_finale} alle {orario_desiderato}.")
```

3. Implementazione degli Algoritmi A* e Dijkstra Modificato:

Abbiamo integrato gli algoritmi A* e Dijkstra Modificato nell'applicativo per il calcolo dei percorsi minimi. Questi algoritmi sono stati adattati per considerare i dati temporali e calcolare i percorsi ottimali in base all'orario specificato dall'utente.

```
#  Implementazione di Dijkstra

import heapq  # Per utilizzare il modulo heapq per l'heap binario

def dijkstra_temporale(grafo, nodo_iniziale, nodo_finale, orario_desiderato):
    # Inizializza le distanze con infinito per tutti i nodi tranne il nodo iniziale
    distanze = {nodo: float('inf') for nodo in grafo}
    distanze[nodo_iniziale] = 0

    # Crea un dizionario per tenere traccia dei predecessori per ciascun nodo
    predecessori = {}

    # Crea un heap binario iniziale con il nodo iniziale
    heap = [(0, nodo_iniziale)]

    while heap:
        # Estrai il nodo con la distanza minima dalla coda
        (distanza_attuale, nodo_attuale) = heapq.heappop(heap)

        # Se abbiamo raggiunto il nodo finale, costruisci il percorso e restituiscilo
        if nodo_attuale == nodo_finale:
            percorso = []
            while nodo_attuale is not None:
                percorso.insert(0, nodo_attuale)
                nodo_attuale = predecessori.get(nodo_attuale)
            return distanza_attuale, percorso

        # Se la distanza attuale è maggiore di quella già calcolata, passa al prossimo nodo
        if distanza_attuale > distanze[nodo_attuale]:
            continue

        # Itera sugli archi uscenti dal nodo attuale
        for vicino, pesi_temporali in grafo[nodo_attuale].items():
            tempo_percorrenza = pesi_temporali.get(orario_desiderato)
            if tempo_percorrenza is not None:
                distanza_alternativa = distanza_attuale + tempo_percorrenza
                # Se la distanza alternativa è più breve, aggiornala
                if distanza_alternativa < distanze[vicino]:
                    distanze[vicino] = distanza_alternativa
```

```
#  Implementazione di A*

coordinate_nodi = {
    'A': (0, 0),
    'B': (1, 2),
    'C': (3, 1),
    'D': (2, 3),
    'E': (4, 2),
    'F': (5, 0)
}

import heapq
import math

def distanza_euclidea(node1, node2):
    # Calcola la distanza euclidea tra due nodi.
    x1, y1 = coordinate_nodi[node1]
    x2, y2 = coordinate_nodi[node2]
    return math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)

def astar_temporale(grafo, nodo_iniziale, nodo_finale, orario_desiderato):
    # Inizializza le distanze con infinito per tutti i nodi tranne il nodo iniziale
    distanze = {nodo: float('inf') for nodo in grafo}
    distanze[nodo_iniziale] = 0
    # Inizializza una coda aperta con il nodo iniziale
    heap = [(0, nodo_iniziale)]
    # Crea un dizionario per tenere traccia dei predecessori per ciascun nodo
    predecessori = {}

    while heap:
        # Estrae il nodo con il punteggio F minimo dalla coda aperta
        _, nodo_attuale = heapq.heappop(heap)
        # Se abbiamo raggiunto il nodo finale, costruisce il percorso e lo restituisce
        if nodo_attuale == nodo_finale:
            percorso = []
            while nodo_attuale is not None:
                percorso.insert(0, nodo_attuale)
                nodo_attuale = predecessori.get(nodo_attuale)
```

4. Risultati della Sperimentazione

Durante il processo di valutazione dell'applicativo, abbiamo ottenuto risultati significativi che riflettono la capacità dell'applicativo di calcolare percorsi ottimali in base alle variazioni temporali. Di seguito, i risultati chiave per alcuni dei percorsi selezionati:

1. Percorso A->D:

- Migliore percorso alle 08:00: A->E->D
- Migliore percorso alle 12:00: A->C->D

Osservando questo percorso sul grafo, è possibile notare che il tratto di strada che ha come partenza *Casa* e come destinazione *Ospedale Maria Vittoria* ha diverse soluzioni, ma i percorsi restituiti dagli algoritmi dimostrano che la variazione dei costi negli orari è significativa.
Difatti, alle ore 08:00, per raggiungere il nodo di destinazione D (ospedale) si preferisce passare per un nodo meno trafficato ovvero il nodo E (supermercato Penny) anziché passare per il nodo C (Scuola), che in quella fascia oraria risulta molto più trafficato.

Differentemente, nella fascia oraria delle 12:00, si predilige il percorso che attraversa il nodo C (scuola) per raggiungere il nodo D (ospedale); ciò avviene perché, cambiando la fascia oraria, la congestione presente alle ore 08:00 relativa al nodo C dovrebbe essersi risolta, non essendo più l'orario di punta per accompagnare i bambini a scuola.

2. Percorso B->D:

- Migliore percorso alle 08:00 e alle 12:00: B->F->D
- Migliore percorso alle 18:00: B->A->C->D

In questa situazione il percorso che abbiamo analizzato è quello che ha come nodo di partenza il supermercato *Conad* e come nodo di destinazione l'*Ospedale Maria Vittoria.*

Si nota che in diverse fasce orarie, ovvero le ore 08:00 e le ore 12:00, il percorso preferito è sempre quello che attraversa il nodo F (Palestra Freestyle SSD) anche se cambiano le condizioni stradali.

Invece, intorno alle 18:00, il percorso che passa per il nodo F non risulta più quello ottimale, in quanto il traffico aumenta e, pertanto, il percorso alternativo migliore risulta essere quello che attraversa il nodo A e il nodo C, rispettivamente *Casa* e *Scuola*.

3. Percorso E->B:

- Migliore percorso alle 08:00 e alle 12:00: E->F->B
- Migliore percorso alle 18:00: E->A->B

Questo percorso si pone l'obiettivo di raggiungere il nodo di destinazione B (supermercato Conad) partendo dal nodo E (supermercato Penny).

Si nota che nelle fasce orarie mattutine (08:00 e 12:00) la situazione stradale non cambia in modo significativo; infatti, il percorso migliore rimane lo stesso.

Questa situazione non si verifica però nelle ore pomeridiane, nelle quali il nodo F (Palestra Freestyle SSD) non risulta essere più il nodo migliore per raggiungere la destinazione. Infatti, si preferisce passare dal nodo A (Casa).

4. Percorso C->F:

- Migliore percorso: C->D->F (per tutti gli orari)

Questo percorso, contraddistinto dal nodo di partenza (scuola) e il nodo di destinazione F (palestra), ci permette di affermare che, indipendentemente dalla fascia oraria selezionata, il percorso ottimale è sempre lo stesso, ovvero il percorso che attraversa il nodo D (ospedale).

5. Conclusioni

Questo studio ci ha permesso di concentrarci su problemi che vengono affrontati quotidianamente:

- le condizioni metereologiche imprevedibili, come piogge improvvise o condizioni di traffico influenzate dal clima, possono rendere difficile pianificare gli spostamenti o le attività quotidiane.
- Con l'aumento dei mezzi di trasporto come automobili, biciclette, mezzi pubblici, servizi di bikesharing e altro ancora, i cittadini possono avere molte opzioni per gli spostamenti. Tuttavia, ciò ha reso le strade sempre più trafficate ed è difficile assicurare a tutti i cittadini corsie adeguate ai relativi mezzi di trasporto.
- La crescente aspettativa di servizi efficienti e tempestivi spinge le città a modificare i loro sistemi di trasporto continuamente, il che tante volte produce un fenomeno opposto a quello desiderato, ovvero il disservizio causato da lavori continui.

Queste situazioni di caos e incertezza possono avere un impatto significativo sul benessere psicologico dei cittadini. L'ansia, lo stress e la frustrazione possono emergere quando le persone si sentono impotenti nel gestire il proprio tempo o quando hanno la sensazione che il tempo scorra troppo rapidamente.
Nel momento in cui le persone perdono la cognizione del tempo, possono sentirsi intrappolate in un ciclo frenetico di attività, senza riuscire a gestire efficacemente i propri impegni.
Il continuo sviluppo di nuove tecnologie sempre più avanzate ha lo scopo di affrontare queste sfide, cercando soluzioni innovative che possano aiutare le persone ogni giorno nelle loro attività.

Questa tesi, quindi, vuole cercare di approfondire il problema della ricerca del percorso migliore in situazioni in cui il tempo è un fattore importante.
I risultati del nostro studio hanno dimostrato che avere un supporto in situazioni di incertezza (come ad esempio quando non si riesce a scegliere la strada migliore per raggiungere un determinato posto) può essere utile per ottimizzare il tempo. Può sembrare banale ma ciò questo può avere effetti positivi sul benessere psicologico del cittadino.

L'applicativo da noi sviluppato è solo il punto di partenza di quella che potrebbe essere un'evoluzione tecnologica. I risultati dimostrano chiaramente la capacità dell'applicativo di adattare i percorsi in base all'orario specificato dall'utente. Le variazioni temporali vengono gestite in modo efficace, consentendo di identificare i percorsi ottimali in diverse fasce orarie.
Naturalmente, uno studio più approfondito potrebbe risolvere le problematiche legate a situazioni di vita reale. Sarebbe, perciò, interessante cercare di rispondere ai nuovi quesiti scaturiti da questo studio, del tipo: "Come si può misurare la congestione stradale?"; "Esiste una metodologia per evitare la congestione stradale?"; "Come si affrontano diversi tipi di congestioni stradali?"; "Come

far fronte a cambi repentini di Condizioni Climatiche?”; “Quali sono le fasce orarie a cui prestare attenzione?”.

In conclusione, questo studio ha l’obiettivo di esplorare il concetto del Time-Dependent Shortest Path nell'ambito delle reti di grafo e delle applicazioni di ottimizzazione, esaminando anche i vari algoritmi volti alla risoluzione del problema.
Inoltre, esso potrebbe rappresentare un punto di partenza per ulteriori ricerche e approfondimenti a tal riguardo.