



UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica e dell'Automazione

Un algoritmo di ottimizzazione per il problema integrato di Timetabling e Vehicle Scheduling

An optimization algorithm for Integrated Timetabling and Vehicle Scheduling

Candidato:
Lorenzo Barontini

Relatore:
Prof. Fabrizio Marinelli

Anno Accademico 2021-2022

Indice

Introduzione	3
Concetti chiave	4
Descrizione del problema trattato	6
Metodo risolutivo	10
Risultati Computazionali	16
Conclusioni	22
Bibliografia	23
Appendice: Dettagli implementativi	24
Ringraziamenti	27

Introduzione

Il processo di pianificazione per un servizio di trasporto pubblico ha come fine ultimo il raggiungimento simultaneo di obiettivi in contrasto fra loro: da una parte il soddisfacimento dei bisogni della clientela, assicurando il servizio con mezzi adeguati al territorio e organizzandone le corse in modo da sopperire alle diverse necessità durante il periodo preso in esame; dall'altra la gestione oculata delle risorse, quindi con l'obiettivo di utilizzare il minor numero di mezzi possibile effettuando percorsi che minimizzino i costi per l'azienda (carburante, dipendenti,...).

Questi obiettivi generali sono comuni ad una qualsiasi azienda di trasporti (taxi, treni, aerei) ma è necessario specificare che questa tesi si focalizza sul trasporto pubblico locale per mezzo autobus.

La pianificazione di un sistema di trasporto pubblico è generalmente suddivisa in fasi che vengono poi trattate in sequenza: Design della rete (*Network Design*, ND), Pianificazione Temporale (*TimeTabling*, TT), Pianificazione Veicoli (*Vehicle Scheduling*, VS), Pianificazione del personale (*Crew Scheduling* CS). I primi due passaggi definiscono il servizio che deve essere offerto, ND definisce l'insieme di linee e quanto spesso il servizio deve essere offerto per ogni linea, mentre TT definisce il tempo di arrivo e di partenza di ogni viaggio (*Trip*) per ogni linea, con l'obiettivo di garantire la frequenza di servizio desiderata. Le ultime due fasi invece si concentrano nell'allocazione delle risorse, VS si pone l'obiettivo di collegare trips ad autobus in modo che ogni trip (scelto in TT) sia percorso da esattamente un bus e che la pianificazione dei veicoli sia fattibile. CS, invece, ha la funzione di collegare trips e personale, in modo che ogni trip (scelto in TT) sia collegato ad esattamente un dipendente e che la pianificazione del personale soddisfi le restrizioni logistiche e legali del caso.

Data la parziale indipendenza delle varie fasi, eseguirle in successione può generare soluzioni sub-ottimali. Questo è particolarmente vero per i veicoli ed i conducenti necessari, in quanto sono determinati solo nelle fasi finali del processo di pianificazione. Sfortunatamente, però, decomporre in fasi il processo risulta necessario per generare una soluzione nei tempi richiesti da chi esegue la pianificazione.

Poichè è possibile definire le fasi di TT e di VS come centrali al processo decisionale risulta interessante un approccio che si concentri nell'integrazione delle due fasi. In questa tesi viene, quindi, proposto un algoritmo di risoluzione del problema Integrato di Timetabling e Vehicle Scheduling (ITTVS) mediante la ricerca di cammini minimi in un grafo di compatibilità. Al fine di risolvere un problema di interesse applicativo e non puramente teorico le istanze di input e la descrizione del problema provengono dalla MINOA Research Challenge [3] che, seppur non collegando il problema ad un caso reale, pone vincoli ed obiettivi realistici.

Concetti chiave

In questa sezione si introducono concetti e nozioni utili alla comprensione del problema.

In prima istanza vengono descritti i problemi di TT e VS in generale, viene poi introdotto il problema di cammino minimo, utilizzato nella ricerca di una soluzione del problema integrato di TimeTabling e Vehicle Scheduling.

TimeTabling

Il processo di creazione della timetable consiste nel creare una programmazione oraria partendo dalla rete di linee e dalla frequenza del servizio desiderata. Il risultato sarà un insieme di viaggi (trips) con il rispettivo orario di partenza e arrivo ai terminal e nelle fermate più importanti delle linee.

Il problema di Timetabling può essere *periodico* o *non periodico*: se l'ordine degli eventi è fissato il problema è non periodico e può essere efficientemente risolto tramite tecniche di cammino minimo; se invece gli eventi si ripetono periodicamente non è possibile ordinarli, motivo per il quale il problema di ordinamento di eventi periodici (periodic event scheduling PESP) è **NP-difficile** [2].

Nel caso di eventi non periodici normalmente si misura l'*headway* di una linea, cioè il tempo che separa due servizi successivi alla fermata principale di una linea. Questo parametro definisce quanto spesso un servizio dovrebbe essere offerto ed è l'inverso della frequenza, nell'arco di tempo normalmente considerato nel timetabling, nel caso periodico. Il problema di TimeTabling si pone come obiettivo quello di trovare soluzioni di buona qualità dal punto di vista dell'utilizzatore del servizio di trasporto pubblico, questo potrebbe avere differenti significati. Probabilmente il più semplice è la regolarità del servizio, dove si cerca di trovare una timetable dove i trips hanno esattamente la frequenza o l'*headway* richiesto dalla linea alla quale appartengono, o almeno che la differenza dal valore ideale sia minimizzata. Un altro obiettivo importante, quando si tratta una rete di trasporto pubblico con più di una linea, è la coordinazione dei trasporti, o sincronizzazione, dove si pone l'attenzione nel cercare una programmazione oraria che minimizzi il tempo di trasporto o il tempo di attesa dei passeggeri nelle fermate che connettono linee diverse.

La fase di TT ha un ruolo tattico nella pianificazione di un trasporto pubblico in quanto mira a ottimizzare il servizio per i passeggeri e, come detto precedentemente, è in diretto contrasto con le fasi di VS e di CS che mirano, invece, alla minimizzazione dei costi.

Vehicle scheduling

Se le linee e le timetable sono note, è necessariamente noto anche l'insieme di trips che deve essere effettuato dallo stesso veicolo, cioè la sequenza di tempi di arrivo e partenza ad ogni fermata di una data linea. L'insieme di trips è l'input della fase di VS, che mira a servirli (tipicamente minimizzando il numero di veicoli necessario). È possibile individuare anche altri

parametri utili a giudicare l'ottimalità di una soluzione come i viaggi fuori servizio (*Deadhead*) o altri costi operativi.

La fase di VS gioca un ruolo importante nella pianificazione di una rete di trasporto pubblico essendo la prima fase nella quale l'obiettivo è la minimizzazione del costo. Il problema della pianificazione dei veicoli ha il compito di creare un set ottimale di sequenze di trip successivi, effettuabili da uno stesso veicolo, in modo che ogni trip della timetable sia presente in esattamente una sequenza. Una sequenza di trip assegnata ad un veicolo può servire più linee. Il problema di VS nel caso ci siano più depositi per gli autobus (Multy-depot) è **NP-difficile** [2], mentre nel caso di un solo deposito può essere risolto in tempo polinomiale.

Problema di cammino minimo su grafi

Un grafo diretto e pesato è una tripla $G = (V, E, w)$ con (V, E) grafo diretto e $w: E \rightarrow \mathbb{R}$ funzione peso che associa ad ogni arco un peso (costo) reale.

Dati 2 nodi u, v il problema di cammino minimo si pone l'obiettivo di trovare, se esiste, il cammino di costo minimo che li collega. Tra i vari modi in cui può essere risolto questo problema, quello scelto in questo documento è l'algoritmo di Dijkstra [1], ampiamente trattato in letteratura e di semplice, ma efficace, implementazione.

La complessità computazionale di tale algoritmo si dipende dal numero di nodi e dagli archi che collegano i nodi nel grafo.

Siano $|V|, |E|$ rispettivamente il numero di nodi e di archi di un grafo. l'algoritmo di Dijkstra Ha complessità $O((|V| + |E|) \log_2 |V|)$

Tale algoritmo determina la complessità di tutto l'algoritmo descritto in questa tesi come verrà spiegato in seguito.

Descrizione del problema trattato

Questa sezione descrive in dettaglio la versione trattata del problema ITTVS.

L'input principale del problema integrato di time tabling e vehicle scheduling è una rete di trasporti pubblica (Indicata in seguito come PTN *Public transportation network*). In generale una PTN è descritta da un grafo dove i nodi corrispondono ai terminal o ai depositi, mentre gli archi indicano il transito diretto di veicoli. Data una PTN un servizio pianificato è formato da un insieme L di linee. Una linea $l \in L$ è un percorso bidirezionale AB nella PTN tra due terminal A_l e B_l (terminal di partenza e terminal di arrivo di una linea). Una linea l ha due direzioni dette in entrata (*in-bound*) ed in uscita (*out-bound*) ed indicate rispettivamente come $D_l = \{\overrightarrow{A_l B_l}, \overrightarrow{B_l A_l}\}$. Si indica con $D = \cup_{l \in L} D_l$ l'insieme di tutte le direzioni, similmente, con $N = \cup_{l \in L} \{A_l, B_l\}$ l'insieme di tutti i terminal collegati dalle linee. Per ogni direzione esiste una fermata principale (rappresentata dall'orologio in Figura 1).

La regolarità del servizio è misurata attraverso l'*headway*, cioè l'intervallo di tempo tra due veicoli consecutivi, appartenenti alla stessa linea, passanti per la fermata principale.

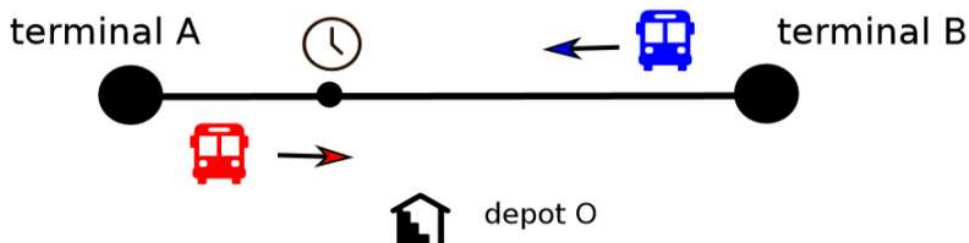


Figura 1, Una linea con la sua fermata principale.

Per ogni linea, per ogni direzione della linea, e per ogni *finestra temporale* in cui l'*orizzonte di tempo* è suddiviso, l'*headway* massimo è dato. Questo definisce il livello di servizio richiesto per la direzione (per la data finestra di tempo) come l'inverso della frequenza minima alla quale i veicoli si fermano alla fermata principale. Le linee sono indipendenti in termini di *headway* massimo, quindi nelle richieste di TimeTabling (TT). Sono, invece, connesse dal fatto di essere servite da un unico insieme di veicoli, cioè dalle richieste di Vehicle Scheduling (VS).

Come la PTN, l'insieme T di *viaggi* potenziali sono specificati nei dati di input. Ogni viaggio $i \in T$ corrisponde ad una direzione univoca $d(i)$ in una linea l nella PTN, è quindi caratterizzato anche da un terminal di *inizio* (start terminal) e di *fine* (end terminal), quelli di $d(i)$, successivamente denotati come $st(i)$ e $en(i)$, con i loro corrispondenti *orari di partenza* (start time) $st(i)$ e di *arrivo* (end time) $et(i)$. È data in input anche la lunghezza in km $l(i)$ del viaggio. Dato che ogni trip appartiene ad una data direzione di una linea, si definisce $T = \{T_d\}_{d \in D}$ come la partizione di direzione di T . È importante notare come tutti i trip in T_d hanno uguale lunghezza ma non hanno stesso tempo di percorrenza, caratteristica fondamentale della configurazione non periodica del problema ITTVS. Infatti, trip nella stessa linea e nella stessa finestra temporale, ma in tempi diversi del giorno, possono avere tempi di percorrenza significativamente diversi, e.g. a causa del traffico nelle ore di punta.

Altra nota importante è data dal fatto che non ogni trip deve essere percorso da un veicolo, in quanto, l'obiettivo del problema ITTVS è proprio quello di selezionare alcuni dei trip

potenziali. Bisogna poi considerare nella PTN, oltre ad i nodi terminal, il singolo nodo deposito O per scopi relativi alla risoluzione del problema VS.

Successivamente sarà indicato con $N^+ = N \cup \{O\}$ l'insieme di tutti i nodi nella PTN rilevanti per il problema.

Vincoli di pianificazione temporale (TimeTabling o TT)

In questa pianificazione non periodica l'orizzonte temporale H è dalle 5:00 alle 27:00 cioè ogni giorno è trattato in maniera indipendente (27:00 rappresenta le 3:00 del giorno successivo). Ogni istante nel tempo è indicato come un intero che rappresenta i secondi (tipicamente ≤ 97200). Per ogni trip $i \in T$ è noto anche il tempo di arrivo alla fermata principale $a(i)$ per la direzione $d(i)$. Tali tempi sono assunti uguali per ogni veicolo che percorre il trip.

Una timetable π_d per una direzione $d \in D$ è un sottoinsieme dei suoi trip potenziali T_d ; una timetable, perciò, è l'unione di $|D|$ timetables indipendenti, una per ogni direzione di ogni linea, cioè $\pi = \bigcup_{d \in D} \pi_d$.

Al fine di misurare la regolarità di una timetable si devono considerare le coppie di *trip consecutivi*; per fare ciò si denota con $P(\pi_d)$ l'insieme di tutte le coppie di trip consecutivi in π_d . Dato un trip i , il suo trip consecutivo j è quello che in π_d passa dalla fermata principale successivamente ad $a(i)$ con minore scarto di tempo, cioè j è quel trip tale che $a(j) \geq a(i)$ e $a(j) - a(i)$ è minimo. Per ogni $(i, j) \in P(\pi_d)$, si definisce headway della coppia come l'ammontare di tempo che separa il loro passaggio nella fermata principale, cioè $w_{ij} = a(j) - a(i)$.

Dato che la frequenza di servizio desiderata tipicamente varia durante la giornata, H è partizionata in k finestre temporali definite da $k + 1$ istanti temporali t_0, \dots, t_k dove t_0 e t_k sono l'istante di tempo iniziale e l'istante di tempo finale di H . Per ogni finestra temporale h ed ogni direzione $d \in D$, è data l'headway massima $I_{d,max}^h$. Per ogni trip i si denota con $h(i)$ la finestra temporale in cui si trova $a(i)$, la finestra temporale è data da $(t_{h(i)-1}, t_{h(i)})$.

Con le definizioni date, una *timetable realizzabile* $\pi_d \subset T_d$ per una direzione $d \in D$ è un insieme di trip che soddisfa tutti i vincoli di headway minima e massima, cioè, tale che: $w_{ij} \leq I_{ij,max} \forall (i, j) \in P(\pi_d)$. Inoltre, il primo e l'ultimo trip di π_d devono appartenere ad un dato sottoinsieme T_d^{ini} e T_d^{fin} di trips iniziali e finali specificati come input del problema.

Vincoli di pianificazione veicoli (vehicle scheduling o VS)

Oltre a percorrere trip in T , i veicoli possono muoversi nella PTN senza passeggeri a bordo, questi viaggi sono chiamati fuori servizio (deadhead trips). In particolare un veicolo che lascia il deposito per raggiungere il nodo di inizio di un trip sta compiendo un viaggio *di estrazione* (*pull-out*), similmente compirà un viaggio *di immissione* (*pull-in*) quando ritorna al deposito dal nodo di fine di un trip.

Per ogni nodo $n \in N^+$ e per ogni finestra temporale h sono dati i tempi minimo e massimo di sosta, denotati rispettivamente con $\delta_{n,min}^h$ e $\delta_{n,max}^h$; infine, si assume che non ci sia un tempo massimo di fermata al deposito. Il periodo nel quale un veicolo è fermo ad un nodo è

detto di *sosta*. È necessario fare distinzione tra tempo di sosta e tempo di *break*: il tempo di sosta è il tempo totale in cui il bus è fermo ad un terminal o al deposito, il tempo di break invece è la porzione del tempo di sosta considerata nella funzione obiettivo della pianificazione veicoli.

Per ogni terminal $n \in N$ e per ogni finestra di tempo h , è dato anche il tempo di *percorrenza* (*travel time*) per i trip di pull-in e di pull-out, denotati con t_{n+}^h e t_{n-}^h , così come le corrispondenti distanze, l_{n+} ed l_{n-} . Si noti che le distanze sono indipendenti dall'orario mentre i tempi di percorrenza lo sono, ma a differenza dei trips il tempo di percorrenza è supposto costante almeno nella medesima finestra temporale.

Due trips $i, j \in T$ non necessariamente appartenenti alla stessa linea sono detti compatibili se possono essere eseguiti consecutivamente dallo stesso veicolo. Questo implica $st(j) \geq et(i)$, cioè, il trip j deve partire dopo che il trip i è finito.

Si distinguono due tipi di compatibilità:

- Compatibilità nella stessa linea cioè:
 1. $en(i) = sn(j)$, il trip j inizia nello stesso terminal nella quale il trip i finisce;
 2. $\delta_{en(i),min}^{h(i)} \leq st(j) - et(i) \leq \delta_{en(i),max}^{h(i)}$, il tempo di sosta al terminal tra la fine del trip i e l'inizio del trip j è realizzabile.
- Compatibilità in linee diverse cioè:
 1. $en(i) = sn(j)$;
 2. $st(j) - et(i) \geq t_{en(i)+}^{h(i)} + \delta_{o+,min}^{h(i)} + t_{sn(j)}^{h(i)}$.

In altre parole, deve esserci abbastanza tempo tra la fine del trip i e l'inizio del trip j per effettuare un trip di pull-in da $en(i)$, aspettare il tempo minimo al deposito ed effettuare un trip di pull-out verso $sn(j)$.

Si noti che non è permesso attendere ai terminal tra un trip ed un pull-in o pull-out, e che i trip deadhead non sono inclusi in T dato che non sono trip di servizio passeggeri.

Nel problema in esame se $en(i) \neq sn(j)$, il veicolo non può muoversi direttamente da un terminal all'altro, ma deve sottostare alle regole di compatibilità tra linee diverse.

Pianificazione dei veicoli con vincolo di realizzabilità

Un *gruppo* (di trips) *realizzabile per un veicolo* è il piano di lavoro per un veicolo per l'intero giorno, composto di un pull-out iniziale, una sequenza di trips compatibili in T , anche separati da break, da pull-in e da pull-out, ed infine un pull-in per ritornare al deposito, tutto condizionato dai vari vincoli.

Un *piano* (di trips) *realizzabile per i veicoli* Ω è un sottoinsieme dei trip potenziali T che possono essere divisi in gruppi realizzabili.

Vincoli di collegamento

Il vincolo per il collegamento tra la parte TT e la parte VS del problema è dato dal fatto che ogni trip nella TT deve essere effettuato da esattamente un veicolo. In altre parole, l'insieme di trips nella TimeTable Realizzabile e nella Vehicle Schedule Realizzabile devono coincidere.

Funzione Obiettivo

L'obiettivo della risoluzione integrata del problema è quello di produrre la soluzione di costo minore al fornitore del servizio. Dato che uno dei costi prevalenti per il fornitore del servizio è il numero di veicoli utilizzati, l'obiettivo primario del problema VS è quello di minimizzare il numero di gruppi realizzabili.

Altri fattori che compongono il costo consistono: nel tempo speso dai veicoli in sosta ai terminal (eccedente il tempo minimo di attesa), nel tempo speso dai veicoli ad effettuare pull-in e pull-out, ed il tempo di guida a causa l'emissione di CO2.

Dato B ad indicare l'insieme di gruppi realizzabili, per ogni $b \in B$ sono definite le seguenti quantità:

1. un costo fisso c_v , per l'uso del veicolo;
2. un costo di break c^{break} , proporzionale al tempo di break t_b^{break} speso ai terminal (non include il tempo minimo di attesa);
3. un costo di pull-in/ pull-out c_v^{pio} proporzionale alla somma dei tempi di pull-in e pull-out t_b^{pi} e t_b^{po} ;
4. un costo per le emissioni di CO2 c_v^{CO2} , proporzionale al tempo di guida $d(b)$.

Quindi in generale la funzione obiettivo è ottenuta come segue;

$$c = \sum_{b \in B} \left(c_v + c^{break} t_b^{break} + c_v^{pio} (t_b^{pi} + t_b^{po}) + c_v^{CO2} d(b) \right)$$

Descrizione della soluzione

La soluzione deve specificare il sottoinsieme $T_s \subset T$ dei trip potenziali che rappresenta sia una Timetable realizzabile sia una Vehicle Schedule realizzabile. T_s deve essere divisa in un insieme di gruppi realizzabili.

Ovviamente è necessario trovare una soluzione di costo minimo.

Metodo risolutivo

Nel seguente paragrafo viene introdotto un approccio per la risoluzione del problema integrato ITTVS utilizzando il grafo di compatibilità e la reiterazione dell'algoritmo di Dijkstra per la risoluzione del sottoproblema di cammino minimo.

Grafo di compatibilità

Al fine di risolvere il problema è possibile rappresentare la PTN attraverso un grafo di compatibilità, cioè un grafo orientato ed aciclico dove i nodi rappresentano i trips potenziali e gli archi rappresentano invece la loro compatibilità.

Serve inoltre aggiungere due nodi: uno di inizio (O^-) che rappresenta l'uscita del veicolo, ad inizio giornata, dal deposito fino al primo terminal di servizio (pull-out trip) ed uno di fine (O^+) che sta ad indicare il rientro in deposito del veicolo a fine giornata (pull-in trip).

Saranno quindi presenti 4 tipi di archi:

- *Archi di compatibilità diretta* (i^+, j^-): per ogni coppia di trip i, j che sono compatibili direttamente (archi verdi in figura 2), se i due trip vengono scelti dall'algoritmo di percorso minimo significa che il veicolo effettuerà il trip i , attenderà al terminal $en(i) = sn(j)$, ed effettuerà il trip j . Il costo di tali archi è proporzionale alla somma del tempo di percorrenza del trip i e del tempo di break.

$$(et(i) - st(j))c_v^{CO2} + (st(j) - et(i) - \delta_{en(i),min}^{h(i)})c^{break}$$

- *Archi di compatibilità indiretta* (i^+, j^-): per ogni coppia di trip i, j che sono compatibili indirettamente (archi rossi in figura 2), se i due trip vengono scelti dall'algoritmo di percorso minimo significa che il veicolo effettuerà il trip i , un pull-in trip al deposito da $en(i)$, un pull-out trip dal deposito verso $sn(j)$, ed effettuerà il trip j . Il costo di tali archi è proporzionale alla somma del tempo necessario ai viaggi di pull-in e pull-out, del tempo totale di guida e del tempo di break al deposito.

$$(t_i^{pi} + t_j^{po})c_v^{pio} + (t_i^{pi} + t_j^{po} + et(i) - st(j))c_v^{CO2} + (st(j) - et(i) - \delta_{dep,min}^{h(i)} - t_i^{pi} - t_j^{po})c^{break}$$

- *Archi iniziali* (O^-, i^-): per ogni trip $i \in T$ (archi neri in figura 2), la scelta di uno di questi archi sta ad indicare che un veicolo effettua la prima attività di un gruppo realizzabile (il primo pull-out trip). Il costo di questi archi è proporzionale alla somma del tempo di pull-out, del tempo di guida e alla differenza tra l'inizio della prima finestra temporale e l'inizio del trip i .

$$t_i^{po}c_v^{pio} + t_i^{po}c_v^{CO2} + (st(i) - st(T))\delta$$

Dove δ è un coefficiente che si può variare per migliorare la soluzione aumentando il costo per selezionare per primi trip che iniziano troppo tardi, di fatto diminuendo la probabilità che l'algoritmo li selezioni (ma lasciandone comunque la possibilità).

- *Archi finali* (i^+, O^-): per ogni trip $i \in T$ (archi azzurri in figura 2), la scelta di uno di questi archi sta ad indicare che un veicolo effettua l'ultima attività di un gruppo realizzabile (l'ultimo pull-in trip). Il costo di questi archi è proporzionale alla somma del tempo di pull-in, del tempo di guida e alla differenza tra la fine del trip i e la fine dell'ultima finestra temporale.

$$t_i^{pi} c_v^{pio} + t_i^{po} c_v^{CO2} + (et(T) - et(i))\delta$$

Dove δ è un coefficiente che si può variare per migliorare la soluzione aumentando il costo per selezionare per ultimi trip che finiscono troppo presto, di fatto diminuendo la probabilità che l'algoritmo li selezioni (ma lasciandone comunque la possibilità).

È importante notare che due trip sono direttamente collegati da massimo un arco e che i costi di ogni arco sono non negativi.

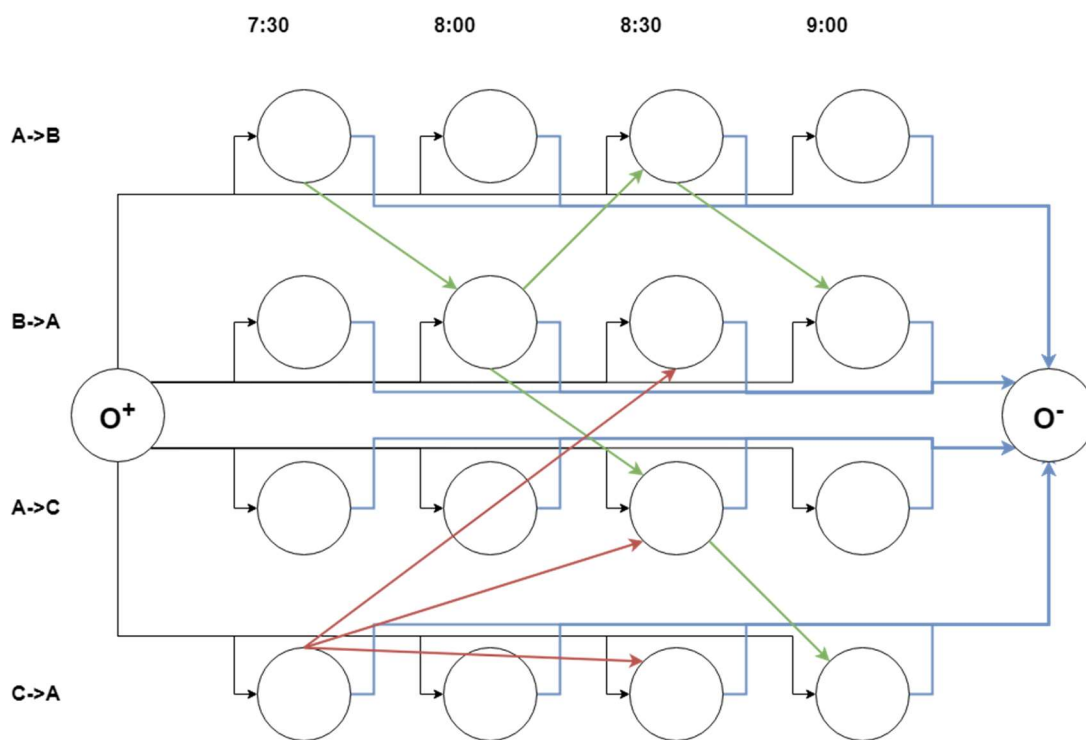


Figura 2 esempio di grafo di compatibilità con i vari tipi di archi che collegano i nodi

Reiterazione dell'algoritmo di cammino minimo

Una volta costruito il grafo di compatibilità, come descritto precedentemente, è possibile trovare un insieme di piani di lavoro per altrettanti autobus che andranno a descrivere la soluzione cercata. È possibile fare ciò cercando, uno dopo l'altro, piani di lavoro di costo minimo. La soluzione scelta è quella di reiterare la ricerca di cammini minimi nel grafo di compatibilità, finché non vengono assecondati tutti i vincoli del problema.

Il costo maggioritario nella funzione obiettivo è rappresentato del numero di autobus utilizzati. Perciò è necessario, come prima cosa, minimizzare il numero di autobus presenti nella soluzione.

In breve, è possibile suddividere la ricerca di una soluzione, dato il grafo di compatibilità, in tre fasi:

- Ricerca di un cammino minimo
- Modifica del grafo
- Analisi dei vincoli

Per comprendere i seguenti passaggi è inoltre utile introdurre il concetto di *prossimità* tra due trips. Due trips potenziali $i, j \in T$ si dicono prossimi se viaggiano dallo stesso terminal di partenza allo stesso terminal di arrivo, stessa linea e stessa direzione, con istanti di arrivo alla fermata principale diversi a meno di un tempo d ; formalmente:

$$\{i, j \in T \mid sn(i) = sn(j), \quad en(i) = en(j), \quad |a(i) - a(j)| \leq d\}$$

Inoltre, data una coppia di trip prossimi $i, j \in T$ si denoterà $mh(i, j)$: la minima headway massima necessaria per la direzione della linea nelle finestre temporali nella quale i trip partono.

Dato che il grafo è diretto, aciclico e con archi di costo non negativo è possibile utilizzare come algoritmo di ricerca del cammino minimo l'algoritmo di Dijkstra, che permette di trovare un percorso minimo in un tempo polinomiale.

Ricerca di un cammino minimo

La prima iterazione dell'algoritmo di Dijkstra viene effettuata con i valori inizialmente assegnati al grafo di compatibilità, mentre successivamente ad ogni iterazione vengono effettuate delle modifiche ai valori di alcuni archi del grafo, poiché saranno resi più convenienti alcuni trip piuttosto che altri, al fine di rispettare tutti i vincoli con meno iterazioni (quindi autobus) possibili.

La modifica dei costi è un processo chiave per la ricerca di soluzioni migliori; perciò, alcuni valori fondamentali sono stati parametrizzati e sarà quindi possibile modificarli per cercare empiricamente una soluzione migliore.

Modifica del grafo

Dato l'insieme di trip scelti dalle iterazioni precedenti dell'algoritmo di cammino minimo si trovano i sottoinsiemi di trip prossimi tra loro, che saranno al massimo in numero pari alla somma di tutte le direzioni di tutte le linee.

Quindi, dato il nodo i appartenente ad uno di questi sottoinsiemi:

Se i è il primo trip scelto per la direzione in esame (con istante di arrivo alla stazione principale minore):

- Si marcano come disconnessi gli archi uscenti da i .
- Se i è initial (figura 3, riga 1):
 - Si marcano e disconnessi gli archi uscenti da tutti i nodi initial nella stessa direzione della stessa linea di i .
- Se i non è initial, serve un altro trip antecedente ad i (figura 3, riga 2):

- Si aumentano i costi degli archi uscenti di tutti i trip j prossimi ed antecedenti ad i tali che $a(i) - mh(i, j)\alpha \leq a(j) < a(i)$ con $\alpha \in \{0,1\}$; ammenoché non siamo Initial.
- Si diminuiscono i costi degli archi uscenti di tutti i trip j prossimi ed antecedenti ad i tali che $a(i) - mh(i, j) \leq a(j) < a(i) - mh(i, j)\alpha$ con $\alpha \in \{0,1\}$.

Sia k prossimo e successivo ad i nel sottoinsieme in esame:

- Se k non esiste:
 - i è l'ultimo trip scelto per la direzione in esame, perciò ci si riconduce al caso in cui i è l'ultimo (descritto successivamente).
- Se $a(k) - a(i) > 2mh(i, k)$, Servono almeno altri 2 trip uno successivo ad i ed uno antecedente a k (figura 3, riga 3):
 - Si aumentano i costi degli archi uscenti di tutti i trip j prossimi e successivi ad i tali che $a(i) < a(j) \leq a(i) + mh(i, k)\alpha$ con $\alpha \in \{0,1\}$.
 - Si aumentano i costi degli archi uscenti di tutti i trip j prossimi ed antecedenti a k tali che $a(k) - mh(i, k)\alpha \leq a(j) < a(k)$ con $\alpha \in \{0,1\}$.
 - Si diminuiscono i costi degli archi uscenti di tutti i trip j prossimi e successivi ad i tali che $a(i) + mh(i, k)\alpha < a(j) \leq a(i) + mh(i, k)$ con $\alpha \in \{0,1\}$.
 - Si diminuiscono i costi degli archi uscenti di tutti i trip j prossimi ed antecedenti a k tali che $a(k) - mh(i, k) \leq a(j) < a(k) - mh(i, k)\alpha$ con $\alpha \in \{0,1\}$.
- Se $mh(i, k) < a(k) - a(i) \leq 2mh(i, k)$, serve un solo altro trip tra i e j (figura 3, riga 4):
 - Si aumentano i costi degli archi uscenti di tutti i trip j , prossimi ad i e k che non rispettano entrambe le headway, cioè tali che $a(j) > a(i) + mh(i, k)$ o $a(j) < a(k) - mh(i, k)$.
 - Si diminuiscono i costi degli archi uscenti di tutti i trip j , prossimi ad i e k che rispettano entrambe le headway, cioè tali che $a(k) - mh(i, k) \leq a(j) \leq a(i) + mh(i, k)$.
- Se $mh(i, k) \geq a(k) - a(i)$, L' headway è rispettata: (figura 3, riga 5)
 - Si disconnettono gli archi uscenti di tutti i trip j , prossimi e compresi tra ad i e k cioè tali che $a(i) < a(j) < a(k)$.

Se k è l'ultimo trip scelto per la direzione in esame (con istante di arrivo alla stazione principale maggiore):

- Si marcano come disconnessi gli archi uscenti da k .
- Se k è final (figura 3, riga 6):
 - Si marcano come disconnessi gli archi uscenti da tutti i nodi final nella stessa direzione della stessa linea di k .
- Se k non è final, serve un altro trip successivo a k (figura 3, riga 7):
 - Si aumentano i costi degli archi uscenti di tutti i trip j prossimi e successivi a k tali che $a(k) < a(j) \leq a(k) + mh(i, k)\alpha$ con $\alpha \in \{0,1\}$.

- Si diminuiscono i costi degli archi uscenti di tutti i trip j prossimi e successivi a k tali che $a(k) + mh(i, k)\alpha < a(j) \leq a(k) + mh(i, k)$ con $\alpha \in \{0,1\}$.

Se k non è l'ultimo:

- pongo $i = k$ e si riparte dall'inizio.

Si effettuano poi questi passaggi per tutti i sottoinsiemi ricavati dalle soluzioni.

Indicando con $|V|$ il numero di nodi del grafo di compatibilità si avrà che la complessità computazionale della fase di modifica del grafo sarà pari a quella della lettura di un vettore di n elementi cioè: $O(|V|)$ quindi trascurabile rispetto alla complessità dell'algoritmo di Dijkstra.

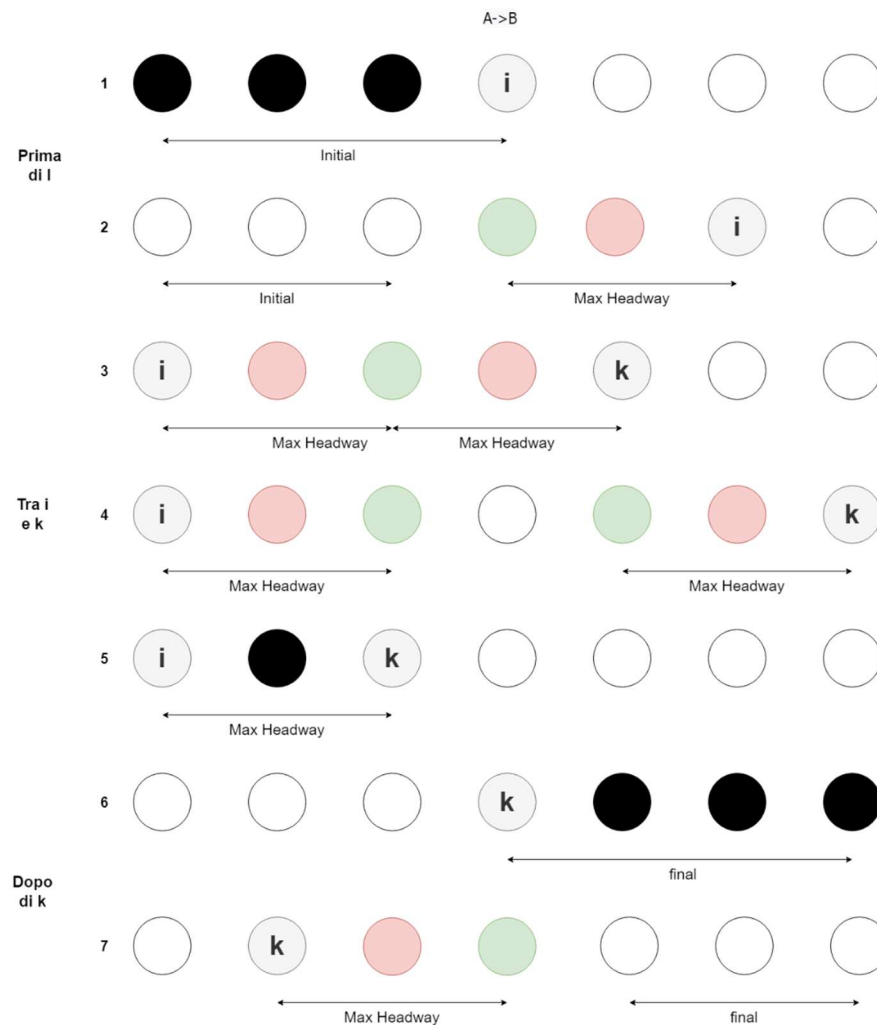


Figura 3; variazione costi archi uscenti dai nodi del grafo di compatibilità nei vari casi. Nero: archi disconnessi, Rosso: costo archi aumentato, Verde: Costo archi diminuito

Aumento e diminuzione dei costi

Come detto in precedenza sono stati introdotti alcuni parametri al fine di permettere un certo grado di regolazione dell'algoritmo, quindi di trovare soluzioni migliori.

Dati due trip prossimi i e j , si devono modificare i costi degli archi uscenti dal nodo j : si moltiplicherà il costo di ogni arco per un coefficiente che dipenderà da un parametro e dal

rapporto tra la differenza di tempo che intercorre tra l'arrivo alla fermata principale dei due trip e la minima headway massima tra quella dei due trip.

Nei casi riportati sopra si avrà che: $|a(j) - a(i)| \leq mh(i, j)$.

In caso di incremento dei costi il coefficiente di moltiplicazione del costo di ogni arco uscente da j sarà: $C_+(i, j) = \beta \left(2 - \frac{|a(j) - a(i)|}{mh(i, j)} \right)$ con $\beta \geq 1$, si nota che tanto più i due trip sono distanti nel tempo, maggiore sarà l'incremento di costo.

In caso di decremento di costi il coefficiente di moltiplicazione del costo di ogni arco uscente da j sarà: $C_-(i, j) = \gamma \left(1 - \frac{|a(j) - a(i)|}{mh(i, j)} \right)$ con $\gamma > 0$, dove una maggiore distanza nel tempo tra i 2 trip farà diminuire il coefficiente.

Variando i parametri β e γ si modifica l'incidenza delle operazioni di incremento e decremento.

Analisi dei vincoli

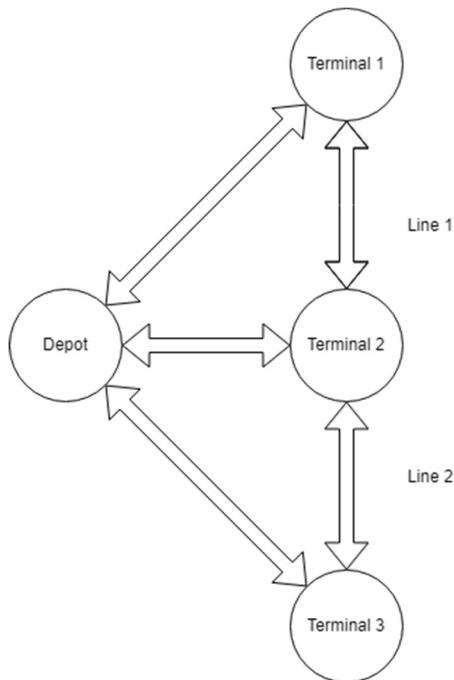
Successivamente l'esecuzione dell'algoritmo di Dijkstra, si controlla se i vincoli sono rispettati per tutte le direzioni di tutte le linee, in caso negativo si procede alla modifica del grafo e si dà inizio ad una nova iterazione dell'algoritmo. In caso positivo l'insieme dei trip scelti da tutte le iterazioni precedenti forma una soluzione al problema ITTVS.

Si può notare che la reiterazione non darà mai origine ad un ciclo infinito in quanto ad ogni iterazione verranno disconnessi tutti gli archi uscenti da almeno un nodo.

Risultati computazionali

Di seguito sono illustrati i risultati trovati con varie istanze di input con diverso numero e configurazione di terminal e direzioni.

2 linee



In questa istanza di input sono presenti 3 terminali collegati da 2 linee, quindi 4 direzioni oltre ai collegamenti al depot.

Variazione alpha

headway media linea 1 inbound: 853.433

headway media linea 1 outbound: 839.118

headway media linea 2 inbound: 865.455

headway media linea 2 outbound: 839.118

Risultato trovato con 16 iterazioni Con un costo totale di 123301760

alpha = 0.7; beta = 2; gamma = 0.2; delta = 10.

headway media linea 1 inbound: 853.433

headway media linea 1 outbound: 839.118

headway media linea 2 inbound: 906.667

headway media linea 2 outbound: 877.846

Risultato trovato con 15 iterazioni Con un costo totale di 115593240

alpha = 0.75; beta = 2; gamma = 0.2; delta = 10.

headway media linea 1 inbound: 853.433

headway media linea 1 outbound: 839.118

headway media linea 2 inbound: 892.5

headway media linea 2 outbound: 877.846

Risultato trovato con 16 iterazioni Con un costo totale di 123299840

alpha = 0.8; beta = 2; gamma = 0.2; delta = 10.

Variazione beta

headway media linea 1 inbound: 866.364

headway media linea 1 outbound: 864.545

headway media linea 2 inbound: 921.29

headway media linea 2 outbound: 920.323

Risultato trovato con 12 iterazioni Con un costo totale di 92472576

alpha = 0.75; beta = 3; gamma = 0.2; delta = 10.

headway media linea 1 inbound: 866.364

headway media linea 1 outbound: 864.545

headway media linea 2 inbound: 921.29

headway media linea 2 outbound: 920.323

Risultato trovato con 12 iterazioni Con un costo totale di 92472576

alpha = 0.75; beta = 4; gamma = 0.2; delta = 10.

Variazione gamma

headway media linea 1 inbound: 866.364

headway media linea 1 outbound: 864.545

headway media linea 2 inbound: 865.455

headway media linea 2 outbound: 864.545

Risultato trovato con 17 iterazioni Con un costo totale di 131006080

alpha = 0.75; beta = 3; gamma = 0.1; delta = 10.

headway media linea 1 inbound: 866.364

headway media linea 1 outbound: 864.545

headway media linea 2 inbound: 906.667

headway media linea 2 outbound: 920.323

Risultato trovato con 9 iterazioni Con un costo totale di 69354648

alpha = 0.75; beta = 3; gamma = 0.3; delta = 10.

headway media linea 1 inbound: 866.364

headway media linea 1 outbound: 864.545

headway media linea 2 inbound: 921.29

headway media linea 2 outbound: 924.194

Risultato trovato con 9 iterazioni Con un costo totale di 69354432

alpha = 0.75; beta = 3; gamma = 0.4; delta = 10.

headway media linea 1 inbound: 866.364

headway media linea 1 outbound: 864.545

headway media linea 2 inbound: 921.29

headway media linea 2 outbound: 924.194

Risultato trovato con 9 iterazioni Con un costo totale di 69354432

alpha = 0.75; beta = 3; gamma = 0.5; delta = 10.

headway media linea 1 inbound: 853.433

headway media linea 1 outbound: 864.545

headway media linea 2 inbound: 906.667

headway media linea 2 outbound: 924.194

Risultato trovato con 9 iterazioni Con un costo totale di 69354864

alpha = 0.75; beta = 3; gamma = 0.8; delta = 10.

Variazione delta

headway media linea 1 inbound: 866.364

headway media linea 1 outbound: 864.545

headway media linea 2 inbound: 921.29

headway media linea 2 outbound: 924.194

Risultato trovato con 11 iterazioni Con un costo totale di 84766528

alpha = 0.75; beta = 3; gamma = 0.5; delta = 5.

headway media linea 1 inbound: 866.364

headway media linea 1 outbound: 864.545

headway media linea 2 inbound: 906.667

headway media linea 2 outbound: 920.323

Risultato trovato con 10 iterazioni Con un costo totale di 77060720

alpha = 0.75; beta = 3; gamma = 0.5; delta = 7.

headway media linea 1 inbound: 853.433

headway media linea 1 inbound: 851.642

headway media linea 2 inbound: 906.667

headway media linea 2 inbound: 920.323

Risultato trovato con 10 iterazioni Con un costo totale di 77061200

alpha = 0.75; beta = 3; gamma = 0.5; delta = 12.

headway media linea 1 inbound: 840.882

headway media linea 1 outbound: 839.118

headway media linea 2 inbound: 906.667

headway media linea 2 outbound: 909.524

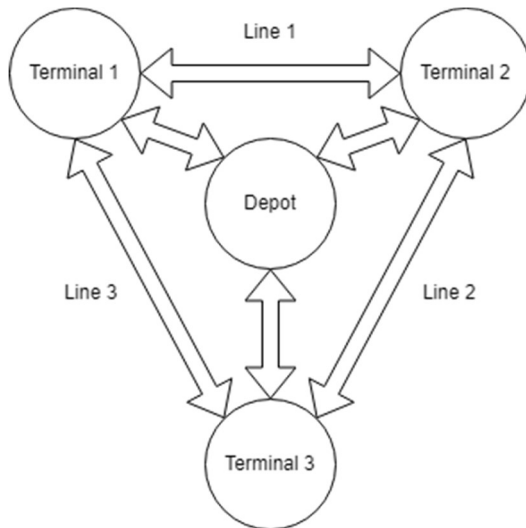
Risultato trovato con 11 iterazioni Con un costo totale di 84768112

alpha = 0.75; beta = 3; gamma = 0.5; delta = 15.

I parametri ottimali tra quelli testati per questa istanza di input sono alpha = 0.75; beta = 3; gamma = 0.5; delta = 10. Quindi, con 9 iterazioni dell'algoritmo (con 9 Bus) la soluzione risultante rispetta tutti i vincoli con un costo totale di 69354432.

Per questa istanza di input è stata presentata anche l'headway media per ogni direzione di ogni linea, tale dato è strettamente correlato al numero di iterazioni e perciò è stato omesso nelle successive istanze prese in esame.

3 linee triangolo



In questa istanza di input sono presenti 3 terminal collegati da 3 linee quindi 6 direzioni oltre ai collegamenti al depot.

Variazione alpha

Risultato trovato con 19 iterazioni Con un costo totale di 146474192

alpha = 0.7; beta = 2; gamma = 0.2; delta = 10.

Risultato trovato con 23 iterazioni Con un costo totale di 177310312

alpha = 0.8; beta = 2; gamma = 0.2; delta = 10.

Risultato trovato con 19 iterazioni Con un costo totale di 146475560

alpha = 0.6; beta = 2; gamma = 0.2; delta = 10.

Variazione beta

Risultato trovato con 27 iterazioni Con un costo totale di 208164384

alpha = 0.7; beta = 1; gamma = 0.2; delta = 10.

Risultato trovato con 24 iterazioni Con un costo totale di 185017152

alpha = 0.7; beta = 3; gamma = 0.2; delta = 10.

Variazione gamma

Risultato trovato con 12 iterazioni Con un costo totale di 92507424

alpha = 0.7; beta = 2; gamma = 0.4; delta = 10.

Risultato trovato con 12 iterazioni Con un costo totale di 92507424

alpha = 0.7; beta = 2; gamma = 0.5; delta = 10.

Risultato trovato con 15 iterazioni Con un costo totale di 115637520

alpha = 0.7; beta = 2; gamma = 0.7; delta = 10.

Variazione delta

Risultato trovato con 15 iterazioni Con un costo totale di 115636080

alpha = 0.7; beta = 2; gamma = 0.5; delta = 5.

Risultato trovato con 13 iterazioni Con un costo totale di 100217624

alpha = 0.7; beta = 2; gamma = 0.5; delta = 15.

Risultato trovato con 13 iterazioni Con un costo totale di 100217624

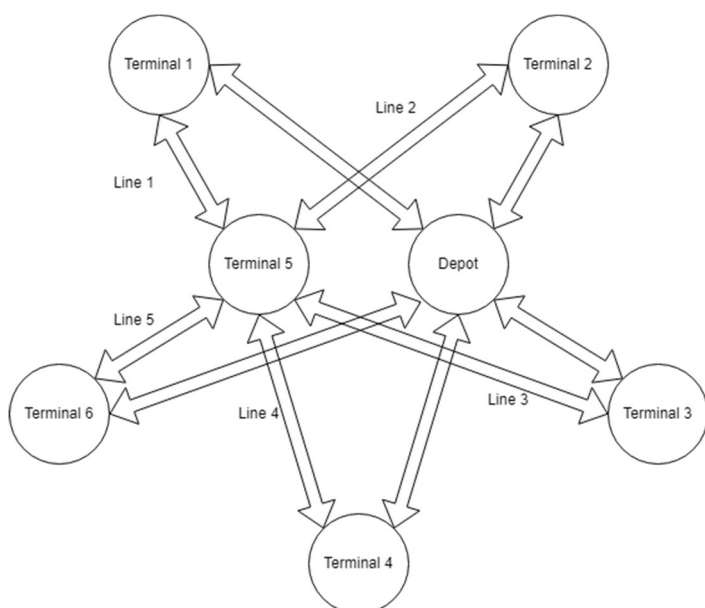
alpha = 0.7; beta = 2; gamma = 0.5; delta = 13.

Risultato trovato con 12 iterazioni Con un costo totale di 92508000

alpha = 0.7; beta = 2; gamma = 0.5; delta = 7.

I parametri ottimali tra quelli testati per questa istanza di input sono alpha = 0.7; beta = 2; gamma = 0.5; delta = 10. Quindi, con 12 iterazioni dell'algoritmo (con 12 Bus) la soluzione risultante rispetta tutti i vincoli con un costo totale di 92507424.

5 linee



In questa istanza di input sono presenti 5 linee quindi 10 direzioni oltre ai collegamenti al depot.

Variazione alpha

Risultato trovato con 21 iterazioni Con un costo totale di 161856744

alpha = 0.65 beta = 3 gamma = 0.5 delta = 10

Risultato trovato con 22 iterazioni Con un costo totale di 169563680

alpha = 0.75; beta = 3; gamma = 0.5; delta = 10.

Risultato trovato con 21 iterazioni Con un costo totale di 161855736

alpha = 0.85; beta = 3; gamma = 0.5; delta = 10.

Variazione beta

Risultato trovato con 36 iterazioni Con un costo totale di 277489440

alpha = 0.85; beta = 2; gamma = 0.5; delta = 10.

Risultato trovato con 18 iterazioni Con un costo totale di 138729600

alpha = 0.85; beta = 4; gamma = 0.5; delta = 10.

Risultato trovato con 18 iterazioni Con un costo totale di 138729600

alpha = 0.85; beta = 5; gamma = 0.5; delta = 10.

Variazione gamma

Risultato trovato con 30 iterazioni Con un costo totale di 231216000

alpha = 0.85; beta = 4; gamma = 0.3; delta = 10.

Risultato trovato con 18 iterazioni Con un costo totale di 138729600

alpha = 0.85; beta = 4; gamma = 0.7; delta = 10.

Risultato trovato con 17 iterazioni Con un costo totale di 131019544

alpha = 0.85; beta = 4; gamma = 0.9; delta = 10.

Risultato trovato con 17 iterazioni Con un costo totale di 131019544

alpha = 0.85; beta = 4; gamma = 0.95; delta = 10.

Variazione delta

Risultato trovato con 17 iterazioni Con un costo totale di 131019544

alpha = 0.85; beta = 4; gamma = 0.9; delta = 5.

Risultato trovato con 18 iterazioni Con un costo totale di 138732624

alpha = 0.85; beta = 4; gamma = 0.9; delta = 15.

Risultato trovato con 36 iterazioni Con un costo totale di 277447968

alpha = 0.85; beta = 4; gamma = 0.9; delta = 1.

Risultato trovato con 18 iterazioni Con un costo totale di 138727008

alpha = 0.85; beta = 4; gamma = 0.9; delta = 7.

Risultato trovato con 17 iterazioni Con un costo totale di 131019952

alpha = 0.85; beta = 4; gamma = 0.9; delta = 9.

Risultato trovato con 18 iterazioni Con un costo totale di 138727008

alpha = 0.85; beta = 4; gamma = 0.9; delta = 6.

I parametri ottimali tra quelli testati per questa istanza di input sono alpha = 0.85; beta = 4; gamma = 0.9; delta = 10. Quindi, con 17 iterazioni dell'algoritmo (con 17 Bus) la soluzione risultante rispetta tutti i vincoli con un costo totale di 131019544.

Conclusioni

L'utilizzo dell'algoritmo descritto in questo documento genera con successo soluzioni al problema integrato di time Tabling e vehicle scheduling non periodico.

Nel capitolo precedente, "Risultati computazionali", la soluzione viene affinata modificando i parametri secondo la sequenza nella quale sono stati introdotti. Si trovano in evidenza i valori ottimali, tra quelli testati, dei parametri nella prima iterazione dell'algoritmo con suddetti valori.

Tale metodo comporta che la soluzione trovata, per quanto buona, può non essere ottima in quanto non sono state esplorate tutte le possibili combinazioni dei valori assegnabili ai vari parametri, è quindi possibile incorrere in soluzioni di minimo locale. Uno studio più specifico della soluzione in base alla variazione dei parametri è consigliabile, ma esula dagli scopi di questo documento.

Bibliografia

- [1] E.W. Dijkstra, "A note on two problems in connexion with graphs", Numerische Mathematik, vol.1, pp.269-271, 1959.
- [2] S. Carosi, A. Frangioni, L.Galli, L.Girardi, G. Vallese, "A Mathematical for Integrated Timetabling and Vehicle Scheduling"
- [3] Minoa Research Challenge https://minoa-itn.fau.de/?page_id=921
- [4] ORTools <https://developers.google.com/optimization/install>
- [5] Nlohmann, Json library (<https://github.com/nlohmann/json>)

Appendice: Dettagli implementativi

Al fine di implementare l'algoritmo descritto sopra si è utilizzato il C++, con delle librerie di supporto quali OrTools[4] dove è implementata la logica che definisce il grafo e l'algoritmo di Dijkstra e, poiché i dati in input sono json, è stato conveniente utilizzare una libreria [5] per trasformarli in oggetti C++.

I metodi più interessanti e fulcro dell'algoritmo sono quelli che vanno a modificare il costo degli archi dato in input il vettore di trip selezionati dalle iterazioni precedenti.

```
int CompGraph::modifyGraph(std::vector<std::vector<int>> resultV, std::vector<model::Direction> directions)
{
    arcCostsReset();

    std::vector<std::vector<NodeAnnotation>> nodesSortedByDir = sortNodesByLineDirection(resultV, directions);
    int count = 0;

    for (std::vector<NodeAnnotation> dir : nodesSortedByDir)
    {
        for (NodeAnnotation node : dir)
        {
            int tripPos = findTripPosition(node);
            if (!g_.IsValid(tripPos))
                throw std::runtime_error("invalid node");
            disconnectNode(tripPos);
            count++;
        }
        count += changeLineDirectionArcsCost(dir);
    }

    return count;
}
```

Nel metodo `modifyGraph`, si divide il vettore risultante dalle iterazioni precedenti per ogni direzione di ogni linea e lo si ordina in base all'istante di arrivo alla stazione principale. Si disconnettono i nodi del grafo in tale vettore e si passa, per ogni direzione di ogni linea, il vettore ordinato di trip al metodo `changeLineDirectionArcsCost`. L'intero risultante dal metodo restituisce il numero totale di nodi disconnessi o ai cui archi è stato modificato il costo.


```

int CompGraph::changeLineDirectionArcsCost(std::vector<NodeAnnotation> dirSortedNodes)
{
    if (dirSortedNodes.size() == 0)
        return 0;
    NodeAnnotation trip = dirSortedNodes[0];

    int modifiedCount = 0, trip_pos = findTripPosition(trip);
    disconnectByInitialFinal(trip_pos);

    if (!trip.isInitial || verbose)
    {
        //increment the cost of the nodes before selected trip to alpha headway unless they are initial
        //decrement te cost of the nodes 1-alpha headway

        for (int target_trip_pos = 0; target_trip_pos < trips_.size(); target_trip_pos++) { ... }
    }

    int i;
    if (verbose)
        std::cout << "{X} ";
    for (i = 1; i < dirSortedNodes.size(); i++)
    {
        modifiedCount += changeArcsCostBetweenNodes(dirSortedNodes[i - 1], dirSortedNodes[i]);
        if (verbose)
            std::cout << "{X} ";
    }

    trip = dirSortedNodes[i-1];
    trip_pos = findTripPosition(trip);
    disconnectByInitialFinal(trip_pos);

    if (!trip.isFinal || verbose)
    {
        //increment the cost of the nodes after selected trip to alpha headway unless they are final
        //decrement te cost of the nodes 1-alpha headway and all the final inside the headway

        for (int target_trip_pos = 0; target_trip_pos < trips_.size(); target_trip_pos++) { ... }
    }

    if(verbose)
        std::cout <<std::endl << std::endl;
    return modifiedCount;
}

```

Il metodo `changeLineDirectionArcsCost` prende in input un vettore di trips selezionati e prossimi tra loro. Se sono presenti trips initial o final: disconnette tutti gli initial o i final nella stessa direzione della stessa linea.

Inoltre, modifica i costi dei trip antecedenti il primo trip del vettore in input, e successivi all'ultimo trip del vettore in input, a meno che non siano initial o final. Infine, invoca il metodo `changeArcsCostsBetweenNodes`, per ogni coppia di nodi immediatamente successiva nel vettore di input.

```

int CompGraph::changeArcsCostBetweenNodes(NodeAnnotation trip1, NodeAnnotation trip2)
{
    int modifiedCount = 0;
    int trip1_pos = findTripPosition(trip1), trip2_pos = findTripPosition(trip2);

    int maxHeadway;
    if (trip1.maxHeadway < trip2.maxHeadway)
        maxHeadway = trip1.maxHeadway;
    else
        maxHeadway = trip2.maxHeadway;

    if (mainstopArrivalTimeDifference(trip1_pos, trip2_pos) > 2* maxHeadway)
    {
        //if the two trips are far apart
        //increment and decrement after trip1
        //decrement increment before trip2
        for (int target_trip_pos = 0; target_trip_pos < trips_.size(); target_trip_pos++) { ... }
    }
    else if (mainstopArrivalTimeDifference(trip1_pos, trip2_pos) <= maxHeadway)
    {
        //disconnect all the nodes between the two since the headway is respected an no further nodes are needed
        for (int target_trip_pos = 0; target_trip_pos < trips_.size(); target_trip_pos++) { ... }
    }
    else
    {
        //decrease the cost of all the nodes that respect the two headways increase the others
        for (int target_trip_pos = 0; target_trip_pos < trips_.size(); target_trip_pos++) { ... }
    }
    return modifiedCount;
}

```

Il metodo changeArcsCostsBetweenNodes riceve in input 2 trips prossimi e modifica i costi degli archi uscenti da tutti i trips, nel grafo di compatibilità, prossimi e con istante di arrivo alla stazione principale compreso tra quello dei due trip in input.

```

//rounding factor for time cost since the DijkstraShortestPath uses int asarc costs
const int COST_ROUNDING = 100000;
const unsigned int DISCONNECTED_DISTANCE = 212121;
const double ALPHA = 0.8; //a quanti trip aumento il costo all interno del max headway
const double BETA = 3; //moltiplicatore aumento costo 1<beta
const double GAMMA = 0.5; //moltiplicatore diminuzione costo 0<gamma<1
const double DELTA = 5; //moltiplicatore costo archi via_depot

```

I parametri sono stati dichiarati come costanti all'interno del documento.

Ringraziamenti

Questa tesi è il risultato di un percorso lungo e burrascoso in cui sono stato accompagnato e sostenuto da molte persone a cui indirizzo questi ringraziamenti.

Alla mia compagna che ha sopperito all'impossibile compito di condividere con me il tetto durante tutti i momenti difficili in questo percorso, dedico non solo un Grazie, ma anche la mia felicità.

Alle mie famiglie, quella a cui devo le mie radici e quella a cui la mia compagna deve le sue, dedico un caloroso Grazie per il supporto nelle mie scelte, e per avermi sopportato in questi anni di cambiamento e crescita.

A tutti i miei amici, per aver subito le pessime battute e gli inutili dettagli degli argomenti che mi appassionano, nonché per la loro vicinanza, Grazie.

Voglio anche ringraziare Renato Fabiani per il tempo condiviso nell'algoritmo che è stato poi oggetto di questa tesi.

Infine, Ringrazio il professor Fabrizio Marinelli e l'UNIVPM per avermi dato i mezzi e la possibilità di affrontare questo percorso formativo, ma soprattutto di crescita.

Lorenzo Barontini.