Sii-Mobility

Supporto di Interoperabilità Integrato per i Servizi al Cittadino e alla Pubblica Amministrazione

Trasporti e Mobilità Terrestre, SCN_00112

Deliverable ID: DE2.4a

Titolo: Stato dell'arte nei problemi di percorso ottimo su grafi multimodali, multi-obiettivo.

Data corrente	26-07-2016
Versione (solo il responsabile puo' cambiare versione	0.2
Stato (draft, final)	Finale
Livello di accesso (solo consorzio, pubblico)	Pubblico
WP	2
Natura (report, report e software, report e HW..)	Report
Data di consegna attesa	Giugno 2016
Data di consegna effettiva	Luglio 2016
Referente primario, coordinatore del documento	Fabio Schoen
Contributor	Luca Tigli tigli.luca@gmail.com, <Nome> <Cognome> <email>, <Nome> <Cognome> <email>
Coordinatore responsabile del progetto	Paolo Nesi, UNIFI, paolo.nesi@unifi.it

Sommario

1 Introduzione ed obiettivi

Questo deliverable descrive lo stato dell'arte e sulla ricerca nel campo dei percorsi ottimi. Questo viene sviluppato nell'ottica del contesto di Sii-Mobility ma escludendo tematiche che sono trattate in altri deliverable come deliverable e algoritmi (come da specifica):

- DE2.2a – stato dell'arte e sulla ricerca effettuata nel campo del vehicle routing, M6
    - A06:Algoritmi di instradamento (veicoli e persone), Attività: 2.2.5
- DE2.4a – stato dell'arte nei problemi di percorso ottimo su grafi multimodali, multi-obiettivo, M6
    - A05:Algoritmi di ottimizzazione (percorsi con più fermate, cambi, etc.), Attività: 2.2.4
    - A07:Algoritmi per la produzione di percorsi per flotte merci, Attività: 2.2.5
- DE2.7a – stato dell'arte della ricerca nel campo dei data analytics, M6
    - A03: Algoritmi di ottimizzazione per la produzione suggerimenti per il parcheggio, Attività: 2.2.1
    - A04a: Algoritmi e strumenti per raccolta e computo flussi di persone, Attività: 2.2.1
    - A04b: Algoritmi e strumenti per raccolta e computo flussi di mezzi, Attività: 2.2.1
- DE2.9a - stato dell'arte e sulla ricerca effettuata nel campo del supporto alle decisioni, M6
    - Identificazione di correlazioni inattese
    - Identificazione di condizioni critiche, dashboard, soluzioni di firing

In questo deliverable lo stato dell'arte relativo ad algoritmi identificati in fase di specifica:

- A05: Algoritmi di ottimizzazione (percorsi con più fermate, cambi, etc.), Attività: 2.2.4

Dall'introduzione dei primi mezzi di trasporto ad oggi la mobilità ha guadagnato un ruolo centrale nello sviluppo tecnologico, portando rilevanti benefici in termini di un più efficiente funzionamento del mercato del lavoro e di un incremento della mobilità delle persone. Tale fenomeno ha tuttavia comportato un intenso sfruttamento del territorio, portando ad una crescita delle reti di trasporto in termini di dimensioni e complessità. La diminuzione dei costi relativi ha generato importanti fenomeni quali la globalizzazione dei mercati, determinando un incremento nella domanda di spostamento di merci unito all'esigenza per le aziende di contenere costi e tempi di trasporto. Dalla fine degli anni ottanta, periodo in cui fecero la loro comparsa i primi dispositivi elettronici per la pianificazione, ad oggi, l'accesso alle tecnologie ed in particolare alle infrastrutture per la connessione alla rete, sono diventati strumenti sempre più alla portata di molti. Il trasporto pubblico, e la mobilità dell'individuo in genere, hanno guadagnato nel tempo un ruolo di assoluto interesse nello sviluppo di applicazioni ed algoritmi efficienti.
Le recenti metodologie di pianificazione si articolano su diverse tipologie di algoritmo, ognuna delle quali scelte sulla base di opportuni compromessi in termini di caratteristiche della rete, tempi di risposta, occupazione di memoria, robustezza e qualità delle soluzioni proposte; si noti come quest'ultimo aspetto possa apparire anche secondario al fine di garantire tempi di risposta plausibili in ambito applicativo.

L'obiettivo di questo documento non è quello di presentare uno stato dell'arte completo ed esaustivo. Ma quello di presentare lo stato dell'arte attuale, mettendo in evidenza gli aspetti recenti delle ricerca e dello sviluppo nei vari settori in modo da servire come punto di partenza per lo sviluppo delle ricerche specifiche.

2 Il problema del percorso ottimo

I grafi sono strutture matematiche discrete che permettono di schematizzare una grande varietà di situazioni e di processi e spesso di consentirne l'analisi in termini quantitativi e algoritmici. Per tali ragioni questi oggetti rivestono interesse sia per la matematica che per un'ampia gamma di campi applicativi fra cui la Ricerca Operativa.
In letteratura i grafi, grazie ad una rappresentazione schematica semplice ed intuitiva, vengono utilizzati come struttura matematica di riferimento per tutti i processi di modellazione urbana; costituendo anche, come verrà approfondito successivamente, una base comune per alcuni fra i più interessanti modelli di trasporto pubblico.

Formalmente si definisce grafo una coppia ordinata

$$\mathbf{G = (V, E)}$$

dove **V** rappresenta un insieme generico di vertici, purché di cardinalità finita $|\mathbf{V}| < \infty$, ed **E** un insieme di archi tali che ogni arco rappresenta una connessione elementare fra due vertici del grafo. Agli elementi di **E** si possono pertanto sostituire coppie di elementi in **V**

$$\mathbf{E} \subset \mathbf{V}\times\mathbf{V}$$

che prendono il nome di estremi dell'arco. Due vertici si definiscono adiacenti se esiste un arco $\mathbf{e}\in\mathbf{E}$ che li congiunge.
La definizione di grafo orientato, o digrafo, differisce da quella di grafo per la presenza di coppie ordinate di nodi contraddistinte da un verso di percorrenza.
Un grafo pesato è un grafo in cui ad ogni arco viene associata, tramite opportune funzioni di costo, una tupla di valori numerici; casi con un numero di elementi superiori ad uno rientrano nell'ambito dell'ottimizzazione multi-obiettivo.

Un percorso orientato fra due nodi **s** e **t**, rispettivamente sorgente e destinazione, nel digrafo **G = (V, E)** lo si può rappresentare come una successione finita di vertici, dove ogni coppia consecutiva del cammino individua un arco in **E** che li connette.
Un percorso i cui tutti i vertici risultano distinti prende il nome di cammino semplice.
Il costo associato al cammino **P** si definisce come la somma dei costi relativi a tutti gli archi che lo compongono ed il problema di cammino minimo da un vertice **s** ad uno **t**, rispettivamente sorgente e destinazione, è quello di determinare, fra tutti i cammini possibili, quello dal costo minimo $\mathbf{P}^*$.

Il problema di individuare un cammino minimo su un grafo generico, con un nodo sorgente ed uno destinazione, può, in molti casi, essere formulato come un problema di flusso su reti di costo minimo ottenendo una formulazione equivalente in termini di programmazione lineare. Per reti di trasporto però, data la complessità e la dimensione dei dati, é comunque preferibile utilizzare algoritmi iterativi del tipo *label-method*.
Questi algoritmi, seppur legati alla programmazione lineare, non ne richiedono la conoscenza dei metodi risolutivi e possono essere applicati con considerevoli incrementi anche dal punto di vista dell'efficienza computazionale.
In generale la struttura di queste procedure non tende a determinare un unico cammino, quello fra la sorgente e la destinazione, ma costruiscono durante le iterazioni della procedura un albero dei cammini minimi **T**, ovvero un sottografo di **G**, in cui il nodo sorgente rappresenta la radice della struttura ed i restanti nodi i soli raggiungibili a partire dal nodo iniziale.
Esistono alcune varianti nella formulazione del problema di cammino minimo:

- **Single-source shortest path problem** nel quale si ricercano i cammini minimi da un nodo sorgente **s** verso tutti i nodi del grafo.

- **Single-destination shortest path problem** nel quale si ricercano per ogni nodo del grafo tutti i cammini minimi verso un nodo terminale **t**. Questo può essere ridotto al caso the single-source shortest path problem considerando il grafo orientato con gli archi invertiti.

- **All-pairs shortest path problem** nel quale si ricercano i cammini minimi fra tutte e coppie di nodi **s**, **t** nel grafo.

In letteratura si è ormai consolidata da tempo la categorizzazione degli algoritmi *label-method* in due famiglie: *label-setting* e *label-correcting*.

Entrambe le tipologie sono iterative e permettono di determinare tutti i cammini ottimi da una sorgente verso i restanti nodi del grafo (*Single-source shortest path problem*). Questi differiscono nelle metodologie di aggiornamento delle etichette (che rappresentano stime superiori ai costi dei cammini ottimi), e nei criteri attraverso i quali gli algoritmi convergono alla soluzione.
Il problema di determinare cammini minimi su grafi a pesi non negativi è stato trattato per la prima volta da Dijkstra in [6]; la sua applicazione a grafi con archi di costo negativo tipicamente porta alla determinazione di soluzioni sub-ottimali.
Assumere tali ipotesi appare comunque una scelta più che naturale tenendo conto della necessità di modellare esclusivamente quantità positive come lunghezze fisiche dei tratti stradali, tempi medi di percorrenza, costo per biglietti o pedaggi, numero di trasferimenti ecc..

Per queste ragioni l'algoritmo di Dijkstra rappresenta la procedura classica nella ricerca dei cammini minimi.
L'algoritmo procede attraverso un etichettatura dei nodi del grafo che attribuisce ad ognuno di questi un valore numerico; tali etichette vengono variate durante il corso dell'algoritmo ma, da una certa iterazione in poi, divengono definitive. L'idea è quella di procedere gradualmente fissando ad ogni passo intermedio le etichette da valori temporanei a definitivi e terminare l'algoritmo nel momento in cui tutti i nodi del grafo corrispondono valori permanenti. Questa procedura divide naturalmente i vertici del grafo in due insiemi: uno contenente tutti i nodi con etichette definitive **N** ed un altro contenente i rimanenti nodi con etichette temporanee. Si noti che ogni etichetta associata ad un nodo $\mathbf{v_i}$ rappresenta un limite superiore al costo del cammino minimo fra la sorgente ed il nodo stesso, e che corrisponderà esattamente al costo ottimo nel caso in cui l'etichetta risulta definitiva. Si può dimostrare anche che ad ogni iterazione intermedia le etichette temporanee stimano il cammino minimo dalla sorgente a tale nodo passando esclusivamente da vertici appartenenti all'insieme permanente **N**. Quindi ogni cammino ottimo è caratterizzato dall'avere sotto-cammini che sono ancora cammini ottimi.
L'algoritmo viene inizializzato includendo soltanto il nodo sorgente nell'insieme dei nodi permanenti, con etichetta definitiva di valore nullo, e aggiungendo i restanti nodi all'insieme temporaneo con etichette di valore ∞. Si noti che se un nodo non risulta raggiungibile a partire dalla sorgente, cioè non esiste un cammino minimo che li congiunge, il valore della sua etichetta rimarrà ∞. Ad ogni un passo intermedio dell'algoritmo viene estratto dall'insieme temporaneo il nodo con etichetta minore (Selezione del nodo) ed aggiunto all'insieme permanente **N**; successivamente vengono rilassati gli archi di uscita dal nodo ed aggiornate tutte le etichette temporanee dei vertici adiacenti al nodo selezionato (Aggiornamento delle Etichette). L'algoritmo termina nel momento in cui non ci sono più etichette temporanee **V = N**. Il cammino minino viene ricostruito a partire

dall'albero dei cammini minimi risalendo dal nodo d'interesse fino alla radice dell'albero *T*, corrispondente alla sorgente del problema, seguendo tutti i suoi predecessori. Ovviamente se un nodo non ha predecessori sull'albero significa che questo non è raggiungibile dal nodo iniziale.
Dal punto di vista della complessità computazionale l'algoritmo di Dijkstra determina una soluzione in un costo asintotico polinomiale; si noti comunque che, l'utilizzo di strutture dati efficienti, può abbassare ulteriormente i tempi di esecuzione. La complessità dell'algoritmo, infatti, deriva principalmente dell'operazione di estrazione (quadratica nel numero dei vertici) ed è quindi fortemente influenzata dall'implementazione della coda. Una struttura particolarmente utile per il problema in esame è il cosiddetto heap, una struttura dati che permette di determinare in modo efficiente il minimo tra un insieme di dati numerici, rappresentando i dati stessi mediante una struttura parzialmente ordinata; in pratica però ottimi risultati si ottengono anche con strutture di tipo 4-heap e 8-heap.

Esistono anche altri algoritmi oltre a quello di Dijkstra che, per determinati scenari, garantiscono prestazioni superiori.
Belleman Ford, ad esempio, ha il vantaggio di non gestire code di priorità e di lavorare anche con pesi a valori negativi; quest'ultimo può utilizzare una semplice struttura FIFO che mantiene traccia di tutti i vertici da esplorare ad ogni iterazione successiva. Trattandosi di un algoritmo della famiglia *label-correcting* può esaminare un nodo più volte.
Floyd Warshall, invece, risolve il problema di cammino minimo fra tutte le coppie di vertici nel grafo ed in pratica, per casi particolarmente densi, risulta migliore di una ricerca di Dijkstra lanciata per ogni singolo nodo.

3 Scenario del trasporto urbano

Esistono varie tecniche per ridurre lo spazio di esplorazione nell'algoritmo di Dijkstra, fra le possibili soluzioni, oltre alla *ricerca bidirezionale* sorgente-destinazione, si riportano sinteticamente alcune tecniche *goal-directed* per orientare la ricerca verso particolari regioni del grafo e velocizzare l'esecuzione complessiva degli algoritmi.

- **A*** rappresenta l'algoritmo *goal-directed* classico per il problema di cammino minimo. Quest'ultimo utilizza una funzione potenziale definita sui vertici del grafo che, di fatto, rappresenta un limite inferiore al reale cammino minimo dal vertice stesso fino alla destinazione. **A*** lavora su una versione modificata dell'algoritmo di Dijkstra in cui la priorità di ogni vertice è alterata secondo il valore assunto dalla funzione potenziale in quel nodo; la speranza è quella di esplorare per primi i vertici prossimi al nodo obbiettivo. In particolare, qualora la funzione potenziale risulti un limite inferiore esatto, l'algoritmo tende ad estrarre esclusivamente i vertici lungo il cammino minimo fra la sorgente e la destinazione. Con un'opportuna scelta della funzione potenziale l'algoritmo può lavorare anche in modalità bidirezionale. Nonostante possa apparire naturale la scelta di un'euristica efficiente su problemi di natura geometrica quali le reti stradali, spesso l'introduzione di una semplice distanza euclidea porta a risultati di velocizzazione praticamente inconsistenti.

- L'algoritmo **ALT** combina **A*** con l'utilizzo di *landmarks* e disuguaglianza triangolare al fine di generare buone stime inferiori sulla distanza. Durante una fase di preprocessazione, vengono valutate e memorizzate tutte le distanze reciproche fra un insieme ristretto di vertici scelti (*landmarks*) ed i rimanenti nodi del grafo; grazie a queste, e alle proprietà della disuguaglianza triangolare, in fase di interrogazione viene calcolata una stima ammissibile

per ogni vertice analizzato. La qualità della stima dipende in generale dalla scelta dell'insieme di *landmarks*, in pratica, per problemi su reti stradali, una buona scelta sembra quella di collocare i vertici in modo sparso lungo tutta la frontiera del grafo.
L'algoritmo **ALT** può essere utilizzato anche in forma bidirezionale; per la ricerca in avanti viene utilizzata la funzione di valutazione scelta per l'algoritmo nella sua versione unidirezionale, per la ricerca all'indietro, invece, la funzione utilizzata è una stima della distanza tra la destinazione e la sorgente valutata sul grafo inverso. La versione unidirezionale di questo algoritmo è applicabile anche su modelli dipendenti dal tempo, data la validità dei limiti inferiori delle funzioni, e la sua efficienza dipende fortemente dalla posizione dei *landmarks*. In merito sono state sviluppate diverse strategie euristiche al fine di ottenere una buona qualità della distribuzione (il metodo maggiormente in uso è l'euristica *avoid*), generalmente però si è soliti optare per un buon compromesso tra i tempi di preprocessazione e qualità dei *landmarks*.

- L'idea dei **contenitori geometrici** è quella di valutare in una fase di preprocessazione tutti i cammini minimi fra i vertici del grafo e, per ogni arco che contiene almeno uno di questi, preservare un informazione su tutti i vertici effettivamente raggiungibili lungo questi cammini per quell'arco. Questo insieme di vertici viene approssimato per mezzo di un settore geometrico che tende a circoscrivere tutti gli elementi all'interno del grafo.
  In fase di interrogazione archi associati a contenitori privi del nodo destinazione vengono esclusi con sicurezza dalla ricerca. Un limite pratico a questa metodologia è la risoluzione del problema *All-pairs* su grafi di grandi dimensioni.

- L'idea per gli **Arc Flags** è simile a quella introdotta con i contenitori, in questo caso però viene ricercata un approssimazione più generale rispetto ad una di carattere geometrico. Durante la fase di preprocessazione viene valutata una partizione del grafo in K celle bilanciate (numero simile di elementi) e caratterizzate da un basso numero di vertici sul bordo. Ogni arco mantiene una struttura di k bit; se per un arco passa un cammino minimo verso un vertice contenuto in una cella k-sima il rispettivo flag viene settato ad uno (zero altrimenti).
  Nella fase di interrogazione ogni arco che non contiene cammini verso la cella di destinazione viene escluso dalla ricerca. Una procedura di etichettatura degli archi per una particolare cella prevede, ad esempio, un istanza di Dijkstra inizializzata per tutti vertici sul bordo della cella (ricerca all'indietro), dove tutti gli archi sull'albero dei cammini minimi ottenuti sono archi da etichettare. Questo algoritmo è il più veloce attualmente in uso su reti statiche, generalmente però richiede tempi di preprocessazione estremamente elevati (di recente reso ancora più competitivo dalla crescente capacità di calcolo disponibile).

- L'idea dell'approccio *Contraction Hierarchies* (**CH**) è quello di sfruttare la gerarchia intrinseca della rete stradale utilizzando gli *shortcuts* come strumento di contrazione di nodi. Intuitivamente si vorrebbero aggiungere degli archi al grafo principale in modo tale da escludere, in fase di ricerca e per richieste di cammino particolarmente lunghe, vertici non rilevanti dall'esplorazione. In pratica un vertice *v* viene contratto con l'aggiunta di un arco (*shortcuts*) fra coppie di vertici adiacenti a quest'ultimo, qualora il cammino minimo fra i due risulti unico e passante per *v* (si utilizzano algoritmi Dijkstra-like).
  Durante la fase di preprocessazione l'algoritmo assegna a ciascun vertice una certa priorità che utilizza poi per determinare un preciso ordine di contrazione che generalmente parte dal vertice meno importante a salire.
  Durante la fase di interrogazione l'algoritmo esegue una ricerca bidirezionale sorgente-destinazione espandendosi lungo gli *shortcuts* verso nodi a priorità crescente (si sale verso i

livelli superiori). I tempi di risposta dell'algoritmo risultano fortemente influenzati dalla scelta dell'ordine di contrazione, che generalmente viene valutato combinando un elevato numero di fattori (la scelta ideale sull'ordine di contrazione risulta a sua volta un problema NP-hard).

- I **metodi di separazione** si basano sull'osservazione secondo la quale i grafi modellati sulla rete stradale si prestano naturalmente a processi di scomposizione. L'idea pratica di questi metodi è quella di separare e sintetizzare la dimensione complessiva del grafo in un sotto-grafo di copertura a dimensione ridotta, che mantenga inalterate le distanze originali, ma sul quale risulti più efficiente applicare algoritmi di ricerca.
  Un recente algoritmo **CRP** si struttura proprio su una separazione per archi del grafo originale, e risulta particolarmente adatto a scenari di tipo *reale*, ovvero che considerano vincoli di manovra ed evoluzioni nel tempo dovute, ad esempio, a fenomeni di congestione del traffico. Per questi scenari sono stati proposti nuovi modelli orientati a semplificare la gestione dei vincoli di manovra.

Combinando fra loro tecniche goal-directed e tecniche ti tipo gerarchico si ottengono algoritmi generalmente più efficienti rispetto all'applicazione di quest'ultime individualmente; una prassi diffusa è quella di costruire una gerarchia di livelli ed utilizzare tecniche goal-directed sul nucleo del grafo ottenuto (generalmente il livello più alto della gerarchia).
Da queste idee prendono vita gli algoritmi **REAL** (combinazione di **ALT** e *reach-based*), **Core-ALT** (contrae i nodi che non richiedono troppi *shortcuts* e poi usa **ALT** bidirezionale), **CHASE** (combinazione di una gerarchia basata su contrazioni di nodi e di tecniche **Arc Flags**), **Nodi di transito** & **Arc Flags** ed infine **SHARC** (combinazione di *Shortcuts* e **Arc Flags**).

In particolare la tecnica **SHARC** permette un calcolo unidirezionale molto rapido di cammini minimi su grafi stradali di grandi dimensioni ed è stata estesa anche a grafi dipendenti dal tempo, si rivela dunque un approccio interessante proprio per la sua caratteristica di poter essere utilizzato anche in contesti dove la ricerca bidirezionale è inapplicabile (il nodo di destinazione non è definito su grafi tempo varianti).
Un ulteriore vantaggio di questo tipo di approccio è che può essere utilizzato anche per trattare problemi con interrogazioni multi-obiettivo. Nell'ambito delle applicazioni reali questo problema si può tradurre nella ricerca di cammini ottimi in termini di minimizzazione del numero di trasbordi, minimizzazione dei tempi di percorrenza, minimizzazione dei tempi di attesa, ecc.. ; risulta invece inefficiente per grafi di tipo dinamico proprio a causa degli *Arc Flags*, che devono essere aggiornati ogni volta che il costo di un arco cambia, e che quindi richiedeno tempi di preprocessazione estremamente elevati.

4 Scenario del trasporto pubblico

In questo scenario l'informazione si trova generalmente raccolta su tabelle orarie. Una tabella oraria consiste di un insieme di fermate, un insieme di linee ed un insieme di corse. Una corsa corrisponde ad un mezzo di trasporto che visita un sottoinsieme di fermate, lungo una certa linea, in specifici istanti di tempo.
Ogni corsa a sua volta può essere scomposta in connessioni elementari, che formalmente si rappresentano come tuple contenenti coppie di fermate origine-destinazione e coppie di istanti partenza-arrivo, fra le quali un mezzo di trasporto transita senza ulteriori soste intermedie.

Il primo problema che sorge è la scelta di un modello adeguato in grado di rappresentare correttamente tutta l'informazione contenuta nelle tabelle orarie; dato lo scenario in cui si collocano ed il fatto di conoscere diffusamente il problema di cammino minimo, si è portati in generale alla costruzione di un modello a grafo.
In questo contesto però, a causa della dipendenza intrinseca della rete dal tempo, ad ogni coppia di vertici corrisponderanno non uno ma un insieme di cammini funzione del particolare istante iniziale τ_d considerato. Si noti come per istanti diversi potrebbero variare non solo i tempi di percorrenza, ma anche le strutture stesse degli itinerari.

Più formalmente, in uno scenario variante nel tempo, il peso **c(u,v)** di un arco **(u,v)∈E** è dato da una funzione **f:R→R** chiamata **Travel Time Function** (*TTF*). Questa funzione specifica il tempo **c(u,v) = f(τ)** necessario a raggiungere **v** a partire da **u** lungo l'arco **(u,v)** considerando come istante di partenza **τ**.
La risoluzione di un problema di cammino minino su reti dipendenti dal tempo è estremamente complesso (*NP-hard*); si richiede perciò la formulazione di una serie di ipotesi sulle funzioni di costo al fine di semplificare l'analisi di queste strutture. Una proprietà interessante, che permette di ottenere soluzioni in tempi ragionevoli applicando algoritmi *Dijkstra-like*, è la proprietà chiamata *FIFO*:

$$\forall \tau' > \tau : \tau' + f(\tau') \geq \tau + f(\tau) , \quad \tau', \tau \in R$$

Questa proprietà assicura, compatibilmente a quanto assunto per le reti stradali con l'introduzione di una velocità media, che percorrendo un arco in una certa direzione non risulti possibile, per istanti di tempo successivi a quello considerato **τ**, ottenere tempi di arrivo precedenti a quello ottenuto con quest'ultimo.
Generalmente per questo scenario si tendono a confrontare diverse metodologie di modellazione, e la scelta è spesso funzione del particolare problema affrontato. L'aspetto fondamentale per poter lavorare con algoritmi Dijkstra-like è riuscire a rappresentare le informazioni contenute sulla tabella oraria come grafi pesati.
In letteratura i modelli più utilizzati sono il modello *tempo-espanso* ed il modello *tempo-dipendente*, per i quali è possibile trovare una trattazione più approfondita in [4].

- Nel caso *tempo-dipendente* ogni stazione viene modellata con un vertice nel grafo e se esiste una connessione elementare fra due di queste sulla tabella oraria sarà presente anche un arco fra i rispettivi nodi. I tempi di partenza e di arrivo vengono tradotti da una funzione del tempo, generalmente lineare a tratti. Tale funzione deve rispettare alcune proprietà per garantire consistenza nei risultati. L'idea è che un arco può essere utilizzato soltanto in concomitanza con la disponibilità di un mezzo di trasporto e con un costo associato dipendente dall'istante in cui si decide di attraversarlo. Praticamente questo significa che un arco non viene pesato con un valore costante, ma piuttosto con una funzione di costo valutata di volta in volta per gli istanti di tempo considerati ("on-the-fly"). Questo modello assume che tutte le funzioni di costo risultino compatibili con la proprietà *FIFO* ottenendo, di fatto, un applicazione diretta dei concetti di grafo dipendente dal tempo.

- Nel caso tempo-espanso invece la costruzione del grafo avviene nello spazio e nel tempo. Si osserva infatti che una tabella oraria consiste di eventi discretizzati. Il modello crea un vertice per ogni evento e poi li collega in successione nella direzione del tempo. Di fatto questa rete rappresenta un vista statica di quella precedente. Un arco è presente fra un nodo partenza, per una stazione s_d, ed un nodo arrivo, per una stazione s_a, se esiste una connessione elementare sulla tabella oraria che congiunge le due stazioni. Ci sono anche

archi fra tutti i vertici appartenenti ad una stazione *stay-edge*, che modellano la possibilità di attendere un mezzo di trasporto successivo; prima di essere connessi i vertici vengono ordinati rispetto alle etichette temporali. Il costo di ciascun arco dipendente dalla natura del problema che si intende risolvere, generalmente corrisponde alla differenza fra le etichette temporali dei due nodi.

L'utilizzo di un *modello tempo-dipendente*, limitando la dimensione del grafo, garantisce in media prestazioni migliori da parte degli algoritmi di ricerca. L'utilizzo dei *modelli tempo-espanso* d'altra parte, trattandosi di reti statiche, semplifica lo sviluppo di algoritmi di ricerca.

Esiste anche una terza categoria di modelli chiamati *modelli a frequenza*. In questo caso l'idea è quella di comprimere tutte le tuple di connessioni elementari per un arco per poi ricostruirle all'occorrenza a partire dalla conoscenza di un certo numero di parametri: un istante iniziale (partenza della prima corsa dalla stazione), un intervallo temporale ed una frequenza di passaggio del mezzo.

Algoritmi che non ammettono una fase di preprocessazione sono da considerarsi buoni in scenari dinamici che includono ritardi, cancellazioni e cambi di itinerario. Il problema di determinare un percorso che minimizzi il tempo di arrivo può essere risolto in modo piuttosto diretto introducendo varianti dell'algoritmo di Dijkstra per entrambi i modelli sopra citati.
Nel caso *tempo-espanso* la procedura è inizializzata sul vertice in corrispondenza del primo evento di partenza disponibile successivo all'istante τ considerato, e terminato in corrispondenza dell'estrazione del primo nodo di arrivo appartenete alla stazione di destinazione (*TED*).
Nel caso *tempo-dipendente*, sotto le ipotesi di funzioni *TTF* di tipo *FIFO* e a valori non negativi, si richiede in aggiunta di valutare nella fase di aggiornamento dell'algoritmo una funzione del tempo in sostituzione di un valore statico (*TDD*).

Tecniche di velocizzazione applicate con successo a reti statiche risultano, in pratica, poco efficienti su reti di trasporto pubblico; le ragioni principali sono da ricercare nelle differenze strutturali fra le due reti di trasporto. In una rete di trasporto pubblico ad esempio, dato un nodo, il numero di vertici mediamente appartenenti ad un suo intorno risulta leggermente più alto rispetto ad un equivalente su rete stradale, giustificando parzialmente un rallentamento su approcci di tipo gerarchico.

- I **Transfer Patterns** sono stai pensati appositamente per il trasporto pubblico. È un algoritmo particolarmente impegnativo dal punto di vista dello sforzo computazionale in fase di preprocessazione, in quanto richiede di risolvere un problema All-pairs per tutti gli istanti di tempo in un intervallo specificato (*range-query*).
  Durante la fase di interrogazione, invece, viene lanciato un algoritmo Dijkstra-like su un grafo ridotto (costruito analizzando tutti i *transfer patterns* fra origine e destinazione) dove ogni arco congiungente due stazioni risulta associato ad una connessione diretta valutabile nell'ordine dei nanosecondi. Esistono alcune metodologie mirate a ridurre il carico computazionale della prima fase (la più diffusa sono gli *hub*) ma in pratica nessuna di queste riesce a velocizzarlo considerevolmente.

  Una ricerca recente rivela che i *Transfer Patterns* rappresentano una metodologia estremamente robusta rispetto a reti affette da ritardi (le soluzioni risultano realmente utilizzabili).

Il problema di integrare dati real-time (ritardi, cancellazioni, ecc..) nel problema di pianificazione risulta ancora piuttosto recente in letteratura, alcune varianti di modellazione sono state proposte di recente al fine di semplificare e velocizzare i processi di aggiornamento dei grafi in fase di interrogazione.

5 Pianificazione multi-obiettivo

Un estensione naturale per i problemi di pianificazione su reti di trasporto pubblico è quella di considerare casi di ottimizzazione multi-obiettivo. In queste circostanze è richiesta l'individuazione di un insieme massimale di soluzioni non dominate secondo Pareto (*curva di Pareto*).

Alcune varianti dell'algoritmo di Dijkstra sono in grado di valutare con esattezza tutto l'insieme di Pareto.

- **MLS** (*Multi-Label Setting*) mantiene per ogni vertice un insieme di etichette non dominate rappresentate come tuple di valori assunti dai vari criteri di ottimizzazione. La coda di priorità mantiene le etichette esplorate gestendole, in generale, secondo un ordine di estrazione di tipo lessicografico. Per ogni etichetta estratta vengono valutati gli archi in uscita sul vertice associato ed effettuati degli aggiornamenti per gli insiemi di etichette sui vertici adiacenti; in pratica si eliminano tutti gli elementi dominati in un insieme tramite un confronto con la nuova etichetta generata.
- In alternativa esiste anche una variante *label-correcting* chiamata **MLC** (*Multi-Label Correcting*) che ad ogni passo dell' algoritmo non analizza la sola etichetta estratta ma tutte quelle non dominate associate al vertice scansionato (per questa variante un etichetta può essere analizzata più volte).

Entrambe le procedure esibiscono prestazioni accettabili per insiemi di Pareto particolarmente ristretti. Sfortunatamente la curva di Pareto può contenere un numero esponenziale di soluzioni anche se ristretto a problemi di tipo bi-obiettivo.
Una possibile soluzione è quella di combinare linearmente i vari obbiettivi eseguendo di fatto un ottimizzazione classica. In alternativa per problemi di cammino minimo è sempre possibile rilassare il vincolo di dominanza ed ottenere un insieme approssimato di dimensione polinomiale facilmente calcolabile.

Alcuni casi pratici bi-obiettivo con tempo di arrivo e numero di trasferimenti sono spesso caratterizzati da insiemi di Pareto piuttosto ridotto, per queste circostanze **LD** (Layred Dijkstra) risulta efficiente. L'idea è quella di fissare un limite superiore k al numero di trasferimenti e costruire k repliche del grafo con opportuni archi di salita rispetto ai casi di trasbordo. In questo modo è sufficiente utilizzare un semplice algoritmo di Dijkstra per calcolare tutto l'insieme di Pareto.
Il limite di questo metodo è l'occupazione di memoria su reti a dimensioni elevate.

Un approccio completamente differente che trae spunto dalla *programmazione dinamica* è **RAPTOR** (*Round-bAsed Public Transit Optimized* Route) [5]. L'algoritmo è stato ideato esplicitamente per operare su reti di trasporto pubblico e nella sua versione di base valuta due obiettivi: tempo di arrivo e numero di trasbordi.
Anzichè utilizzare un grafo l'algoritmo organizza i dati in ingresso (linee e corse) su semplici strutture dati come gli *array*, per poi applicare i principi della programmazione dinamica sulle tabelle orarie: lavora per *rounds*, al *round* k vengono determinate tutte le soluzioni ottime con esattamente k trasbordi. Ogni *round* elabora in ingresso le fermate ottenute dall'iterazione

precedente esaminando tutte le linee in relazione a queste (inizializzato con la fermata di partenza). Ogni linea viene percorsa lungo le sue fermate successive considerando la prima corsa disponibile ed aggiornando se necessario le etichette di costo associate alle varie fermate incontrate. L'algoritmo analizza ogni linea esattamente una volta quindi risulta particolarmente efficiente e di almeno un ordine più veloce rispetto a MLS (ben predisposto anche a parellizzazione). Esistono altre varianti di RAPTOR che permettono di ottimizzare diverse combinazioni di criteri.

6 Pianificazione multimodale

L'approccio generale più intuitivo alla modellazione multimodale prevede la generazione indipendente di grafi per ogni modalità di trasporto ed un unione a posteriori tramite opportuni archi di salita.

Esistono diverse formulazioni del problema di pianificazione multimodale: alcune richiedono l'esclusione di specifiche modalità di trasporto altre multi-obiettivo di costruire un insieme di soluzioni di Pareto da fornire all'utente. Per quest'ultima classe di problemi alcuni approcci tendono a combinare linearmente gli obbiettivi al fine di introdurre una preferenza da parte degli utenti, sfortunatamente queste metodologie esplorano soltanto una piccola parte della curva di Pareto.

Una tecnica che valuta l'intero insieme di soluzioni è quella con un **MLS** (o **MLC**) puro che però su reti di grandi dimensioni si caratterizza per problemi sui tempi di risposta ed utilizzo di risorse; per migliorare le prestazioni l'insieme di Pareto può essere approssimato tramite un rilassamento della relazione di dominanza (scelta di un valore di soglia ε opportuno).

In alternativa è stato proposto di recente

- **MCR** (*Multimodal Multicriteria RAPTOR*) [8] costruito sull'utilizzo di RAPTOR e che opera in modo diverso da una combinazione di ottimizzazioni indipendenti sulle singole reti. L'algoritmo è suddiviso ancora per *round* e per ognuno di questi si eseguono due fasi:

  la prima (RAPTOR) analizza la rete di trasporto pubblico la seconda (MLC) lavora sulla rete urbana; le etichette valutate con la prima fase vengono ovviamente utilizzate come ingresso per la seconda. L'algoritmo viene inizializzato lanciando una ricerca di Dijkstra in modo da valutare tutti i cammini minimi verso le fermate accessibili (e verso l'obbiettivo) e costruire un insieme di etichette d'ingresso per MCR.

  Esistono alcune euristiche per velocizzare MCR come ad esempio utilizzare un rilassamento per la relazione di dominanza oppure applicare metodi gerarchici per ridurre la dimensione della rete urbana.

Per l'utilizzo pratico della curva di Pareto può essere necessario definire una fase di postprocessazione che determini le soluzioni più significative da proporre all'utente fra le molte disponibili. Nel lavoro precedente [**Errore. L'origine riferimento non è stata trovata.**] ad esempio si utilizzano con buoni risultati classificatori di tipo fuzzy.

7 Bibliografia

[1]Lori, A., *Ricerca di cammini ottimi multimodali,* Tesi di Laurea Magistrale, Firenze, 2007.

[2]Geisberger, R., *Advanced Route Planning in Transportation Networks*, Tesi di Dottorato, Fakültat für Informatik des Karlsruher Instituts für Technologie, München, 2011.

[3]Nannicini, G., *Point-to-Point Shortest Paths On Dynamic Time-Dependent Road Networks*, Tesi di Dottorato, École Polythecnique, Palaiseau, 2009.

[4]Frank Schulz, Dorothea Wagner, *Using multi-level graphs for timetable information in railway system*, Springer-Verlag, 2003.

[5]Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner and Renato F. Werneck *Route Planning in Transportation Networks 2015*

[6]Dijkstra, E. W., *A note on two problems in connexion with graphs*, Numerische Mathematik 1: 269–271 (1959).

[7]Daniel Delling, Thomas Pajor, Renato F. Werneck, *Round-Based Public Transit Routing* 2014

[8]Daniel Delling, Dorothea Wagner, Renato F. Werneck, *Computing Multimodal Journeys in Practice* 2013

[9] Fabio Schoen, Teoria e metodi di ottimizzazione lineare, 1991.

[10] Bondy, Murty, Graph Theory Springer 2008.

[11] Pyrga Max-Planck-Institut fur Frank Schulz and Dorothea Wagner University of Karlsruhe, Efficient Models for Timetable Information in Public Transportation Systems, ACM Journal of Experimental Algorithmics, Vol. 12, Article No. 2.4, Publication June: 2008.

[12] Matthias MllerHannemann, Yann Disser, Mathias Schnee Multi-criteria Shortest Paths in Time-Dependent Train Networks, Experimental Algorithms, Lecture Notes in Computer Science Volume 5038, 2008, pp 347-36

[13] Dorothea Wagner, Algorithm engineering for route planning: an update, Springer-Verlag 2011.

[14] Daniel Delling and Renato Werneck, Faster Customization of Road Net-works, in Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13), Springer, 2013

8 Acronimi

- ALT : A-star, landmarks and triangle inequality
- CH: Contraction Hierarchies
- REAL: *Reach and ALT*
- SHARC: *Shortcut and Arc Flags*
- CHASE: *CH and Arc Flags*
- CRP: *Customizable Route Plannig*
- TTF: *Time Travel Function*
- TED: *Time-Expanded Dijkstra*
- TDD: *Time-Dependent Dijkstra*
- MLS: *Multi-Lablel Setting*

- MLC: *Multi-Label Correcting*
- MCR: *Multimodal Multicriteria RAPTOR*
- RAPTOR: *Round-bAsed Public Transit Optimized* Route