Ottimizzazione delle frequenze dei trasporti pubblici

Balsamo D., Demaria R., Panigutti C., Sala A.

8 marzo 2016

Figura 1: Rete completa delle linee e delle fermate della GTT estratta da Openstreetmap e visualizzata con QGis

Indice

1 Introduzione

Lo scopo del progetto è ottimizzare le frequenze di passaggio dei bus del Gruppo Trasporti Torinesi (GTT) in modo da bilanciare la soddisfazione del cliente ed i costi del servizio offerto dalla società. Il modello è stato implementato in GAMA secondo il paradigma di modellizzazione ad agenti.

2 Definizione del problema

Il proggetto mira a cercare la frequenza ottimizzata di passaggio dei bus. Gli indicatori utilizzati nel modello per raggiungere il nostro scopo sono la **soddisfazione del cliente** e l'**efficienza del servizio**.

Supponendo che un cliente sia molto soddisfatto se giunge velocemente a destinazione (tenendo conto della distanza che lo separa dal suo target), è stata quindi definita:

$$\boxed{\text{Soddisfazione del cliente} = \frac{\text{Distanza che separa il passeggero dal suo target in linea d'aria}}{\text{Tempo impiegato per raggiungere il target}}}$$

La soddisfazione del cliente ha lo stesso ordine di grandezza di una velocità, quindi viene normalizzata ad 1 dividendola per $50\frac{km}{h}$ che corrisponde alla velocità massima con la quale può essere fatto il percorso (la velocità dei pullman).

Si è inoltre supposto che una linea che trasporti pochi passeggeri in relazione al numero massimo di passeggeri che potrebbero essere trasportati, ovvero la capienza massima del bus, sia *poco efficiente* (compia molti viaggi senza passeggeri) e quindi che comporti una spesa inutile alla società.

È stata quindi definita l'**efficienza di un singolo bus** come:

$$\boxed{\eta_b = \frac{\text{Numero di passeggeri trasportati in un viaggio di A/R dal capolinea}}{2 \cdot \text{Capacità del bus} \cdot \text{N.ro di fermate della linea}}}$$

$$\text{così } 0 \leq \eta_b \leq 1$$

dove il *numero di passeggeri trasportati in un viaggio di A/R dal capolinea* viene calcolato iterativamente come la somma del numero di passeggeri effettivamente trasportati tra ogni coppia di fermate dell'itinerario.

Possiamo dunque calcolare l'**efficienza della rete** come:

$$\boxed{\eta_{\text{TOT}} = \frac{\sum_{\text{b} \in \text{bus del run}} \eta_{\text{b}}}{\text{Numero totale di bus del run}}}$$

$$\text{così } 0 \leq \eta_{\text{TOT}} \leq 1$$

dove il *numero totale di bus del run* a denominatore è il numero totale di bus che hanno circolato nella rete durante una data simulazione (non è detto che cambiando le frequenze di partenza dei bus dai capolinea il numero totale di bus circolanti sia uguale per ogni simulazione).

La variabile che si cerca di ottimizzare è la **frequenza** φ con la quale i bus del modello vengono generati al capolinea, essa viene impostata all'inizio di ogni simulazione e viene tenuta costante per tutta la durata di quest'ultima. Per semplicità si suppone la frequenza φ uguale per tutte le linee.

Per chiarezza specifichiamo che in questa relazione si utilizzerà la parola *frequenza* in maniera improria, infatti viene intesa come il lasso di tempo che intercorre tra una creazione e l'altra dei bus ai capolinea e non come l'inverso di un tempo. Inoltre useremo in maniera intercambiabile il termine *secondi* e *ticks* (per default GAMA 1 $tick = 1s$) che verrà anche considerato l'unità di misura della frequenza.

Da una analisi dati preliminare si è osservato che la soddisfazione del cliente è una funzione monotona decrescente della frequenza di generazione dei bus al capolinea, mentre l'efficienza è monotona crescente della frequenza.
Abbiamo quindi assunto che la **frequenza ottimizzata** φ^* sia quella che rende uguali la soddisfazione del cliente e l'efficienza del servizio.

Il range di frequenze esplorate varia tra una frequenza di generazione dei pullman ai loro capolinea di 5 minuti ad una di 23 minuti a passi di 2 minuti. È stato scelto questo intervallo di frequenze perchè da un'analisi preliminare compiuta su un range di frequenze molto ampio è risultato essere quello contenente l'intersezione della curva della soddisfazione e dell'efficienza.

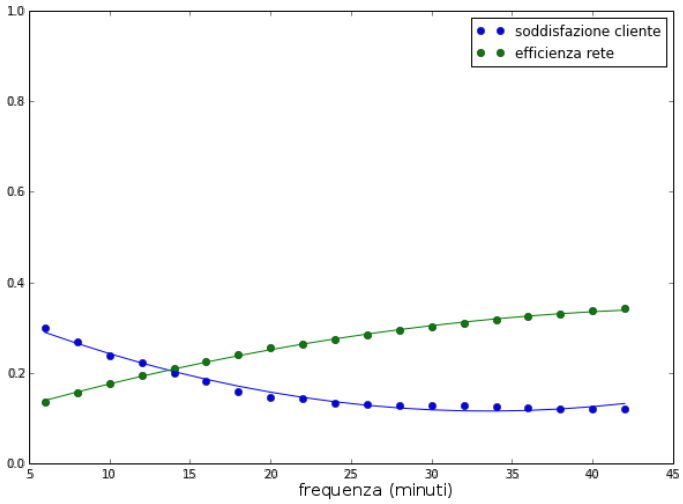

Figura 2: Range espolorato nelle analisi preliminari

3 Descrizione del modello

Il modello è costituito da un ambiente la cui topologia è definita dall'importazione di dati di tipo GIS (Geographic Information System) da uno shapefile che è stato creato a partire da dati opensource. Dentro questo ambiente interagiscono tre tipi di agenti: le **persone**, i **bus** e le **fermate**.

Il modello è stato implementato usando GAMA e quindi utilizzando il linguaggio di programmazione GAML. Esso si sviluppa su due file diversi, il primo chiamato **creapersone.gaml** in cui vengono creati gli shapefile contenenti i dettagli delle persone, meglio descritto nella sezione *Setup del modello* 3.6, il secondo chiamato **run_da_shape.gaml** dove vengono importati gli shape file ed avviene l'effettiva simulazione, meglio descritto nella sezione *Dinamica del modello* 3.7.

L'ambiente di simulazione usato è la rete dei trasporti urbani del Gruppo Torinese Trasporti (Gtt). Per alleggerire il carico computazionale non sono state usate tutte le linee dei bus ma solo una quindicina di quelle che coprono la zona del centro cittadino.

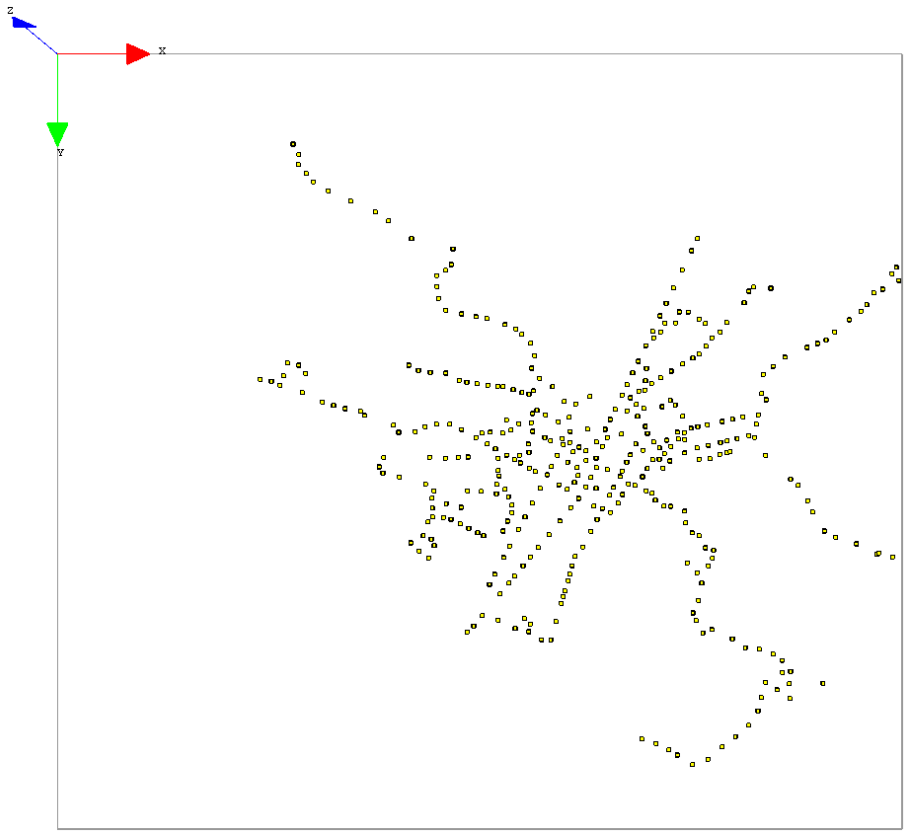

Figura 3: Rete utilizzata in GAMA per le simulazioni

3.1 Topologia dell'ambiente

Allo scopo di ricreare la rete dei trasporti come insieme di fermate e di linee di bus collegate tra loro è stata scaricata la porzione di mappa del comune di Torino fornita dal sito di mappe open source openstreetmap.org in formato GIS (Geographic Information System) grazie all'ausilio degli strumenti di esportazione forniti dal sito mapzen.com,

Utilizzando il programma QGis, sono state quindi estratte le fermate GTT dalla mappa completa (comprendente strade, edifici, etc) e sono state salvate in uno shapefile *FermateGtt.shp* sotto forma di punti georeferenziati, completi delle informazioni loro attinenti quali il nome della fermata e la ref numerica, ovvero il codice identificativo della palina per l'azienda dei trasporti.

Gli agenti di tipo fermata sono stati creati dal file *FermateGtt.shp* ereditando gli attributi di posizione, ref e nome. Dal sito della GTT sono stati inoltre creati dei file .csv contenenti, per ogni linea della rete, le ref delle fermate appartenenti alla stessa linea, ordinate secondo l'ordine reale tra un capolinea e l'altro.

Ad ogni fermata sono stati assegnati una lista *linee* di linee che la attraversano ed un dizionario *ordine* (linea, ordine) che esprime l'ordine cardinale di quella fermata per la linea selezionata. Il dizionario è stato riempito facendo un confronto tra la ref di quella fermata e tutte le ref presenti nei file *.csv* delle linee.

Inoltre per modellizzare in maniera corretta le fermate attraversate da più linee e quei gruppi di fermate formalmente separate ma appartenenti allo stesso cluster, ad esempio tutte le fermate di una piazza, e quindi rappresentabili come un unico nodo di scambio nella rete dei trasporti, viene chiamata l'azione globale *accorpamento_fermate* descritta nella sezione 3.3 che rende queste ultime un'unica fermata della rete.

3.2 Tipi di agenti implementati (entities di GAMA)

In questa sezione vengono elencati gli agenti implementati nel modello. Sebbene alcuni degli attributi degli agenti implementati siano utilizzati solo in uno dei due file del modello, per maggiore chiarezza verranno raccolti e spiegati assieme.

persone : Ogni persona possiede un *id* che la identifica univocamente, una *fermata di partenza* e una *fermata target*. La persona, o passeggero deve arrivare il più velocemente possibile alla propria fermata target facendo al massimo $n_{max} = 3$ cambi. La specie persona è stata definita come entities in GAMA ma non possiede riflessi, quindi non è un propriamente un agente ma un oggetto utilizzato come struttura dati. Possiede due azioni:

- *search_path*: calcola tutti i percorsi che possono portare la persona a destinazione (l'implementazione dell'algoritmo di ricerca del percorso è spiegato dettagliatamente nel paragrafo dedicato 3.4).

- *scegli_percorso*: sceglie quale percorso intraprendere tra quelli possibili, per far questo utilizza una funzione utility (la cui implementazione è spiegata nel dettaglio nel paragrafo dedicato 3.5).
- *salva_su_file_persona*: salva su file di output i dati relativi al viaggio della persona nel momento in cui essa arriva a destinazione.

fermate : Ogni fermata possiede delle coordinate geografiche ed una lista delle linee che la attraversano. Le fermate hanno diversi compiti:

- creare le persone, assegnando loro una *fermata di partenza* e una *fermata target* scelta a random tra tutte le altre fermate della rete di trasporti.
- decidere se la persona è giunta a destinazione o deve aspettare un nuovo bus. Questo compito è implementato nel riflesso (un'*intention* del paradigma BDI) chiamato *nuovi_passeggeri* che si attiva tutte le volte che un bus arriva e scarica dei passeggeri alla fermata. Questo riflesso processa ogni singolo passeggero e decide se aggiungerlo alla lista delle persone che aspettano oppure se è arrivato a destinazione. In questo caso il riflesso salva su file i dati relativi al viaggio di quella persona ed elimina quell'agente. Abbiamo deciso di eliminare gli agenti persona quando questi arrivano a destinazione per snellire la memoria di cui necessita il nostro programma.

Ogni fermata inoltre può compiere due azioni quando un bus arriva alla fermata:

- *carica_passeggeri_su_bus*: questa azione prende come input un'agente della specie bus e verifica quali persone che stanno attendendo alla fermata debbano salire su quel bus.
- *calcola_utility*: questa azione fa calcolare alla fermata qual è il miglior percorso da far prendere ad ogni persona tra i percosi calcolati da *search_path* . L'implementazione di questa azione è meglio descritta nel paragrafo dedicato 3.5.

bus : ogni bus possiede una lista di tutte le fermate che deve attraversare, *lista_fermate*, un capolinea, una capacità massima di passeggeri che può trasportare *bus_capacity* (nel nostro modello settata a 60 per tutti i bus della rete), e una lista dei passaggeri che sta trasportando *lista_passeggeri*.

Gli agenti di tipo bus possiedono la skill *moving* che gli permette di spostarsi tra una fermata e l'altra usando l'azione built-in *goto*; questa azione viene chiamata nel riflesso *spostamento_tra_fermate* che inoltre nel momento in cui il bus arriva ad una fermata chiama l'azione *task_fermata* descritta in seguito.

I bus possono compiere un solo viaggio di andata e ritorno, al termine del quale vengono distrutti.

Le azioni che può svolgere un agente di tipo bus sono le seguenti:

- *crea_lista_fermate*: questa azione viene eseguita una sola volta dopo che il bus è stato creato e riempie la lista delle fermate che il bus deve attraversare e lo posiziona al capolinea.

- *sono_alla_fermata*: questa azione verifica se il bus si trova ad una delle fermate della lista oppure se è in viaggio tra due di esse.
- *controllo_senso_di_marcia*: questa azione verifica se sia il caso di invertire il senso di marcia e nel caso lo inverte.
- *task_fermata*: questa azione esegue tutti i task che deve fare il bus ad ogni fermata, ovvero chiama le azioni *controllo_senso_di_marcia*, *bus_scarica_passeggeri*, *bus_carica_passeggeri*, *check_distruzione_bus*, *seleziona_prossima_fermata* descritte di seguito. Determina inoltre il tempo di attesa del bus alla fermata, calcolato come 2 secondi per ogni passeggero in salita o in discesa; al termine di questo tempo il bus viene autorizzato a partire.
- *check_distruzione_bus*: controlla se il bus è tornato al capolinea di partenza ed in caso affermativo chiama l'azione *salva_su_file_bus* e distrugge il bus.
- *seleziona_prossima_fermata*: questa azione seleziona la prossima fermata tenendo conto della direzione del bus.
- *bus_scarica_passeggeri*: verifica quali dei suoi passeggeri devono scendere alla fermata e li sposta nella *lista_nuovi_passeggeri* di quella fermata.
- *bus_carica_passeggeri*: chiede alla fermata di caricare nella *lista_passeggeri* del bus i passeggeri che devono salire.
- *calcola_ETA*: restituisce il numero di fermate che separano il bus dalla fermata passata come argomento. Questa azione è chiamata dalla fermata all'interno di *calcola_utility*.
- *salva_su_file_bus*: salva su file l'efficienza del viaggio del singolo bus.

3.3 Azioni e riflessi globali

In questa sezione è spiegato il funzionamento delle azioni e dei riflessi globali utilizzati nei file *creapersone.gaml* e *run_da_shape.gaml*.

- *accorpamento_fermate*: azione globale (presente in entrambi i file *.gaml*) che chiede ai gruppi di fermate geograficamente vicine di accorporsi in un'unica fermata, ovvero ogni fermata appartenente alla linea l-esima chiede, alle fermate con una distanza da essa inferiore a *distanza_accorpamento* ed apparteneti ad altre linee, la lista di linee *linee* a cui appartengono, aggiunge questa lista alla propria ed elimina i duplicati

  (es. date le liste di linee fermata1.linee =[10,33,45] e fermata2.linee = [10,42], la 'fermata accorpata' finale dovrà avere come linee [10, 33, 45, 42] e non [10,33,45,10,42]).

  Chiede inoltre a queste fermate il dizionario *ordine* e lo aggiunge al proprio. Infine chiede a queste fermate di morire.

- *salva*: riflesso presente solamente in *creapersone.gaml* che salva in diversi shapefile (ad esempio *salva_persone_1.shp*) i dati delle persone create nel file *creapersone.gaml*, utilizzati successivamente nel file *run_da_shape.gaml*. Fa uno shuffle della lista *persona* di tutte le persone create e la divide in un

numero *n_persone* di partizioni. Successivamente salva in altrettanti shape file l'id, la fermata di partenza, di target e la lista dei cambi possibili di ogni persona creata nelle partizioni (processo meglio spiegato nelle sezioni 3.6, 3.7 e 4).

- *crea_persone*: riflesso presente solamente in *run_da_shape.gaml* che si attiva durante il runtime ogni *spawno_persone* tick (settato a 120 ticks, ovvero 2 minuti). Crea delle persone da uno degli shapefile creati precendentemente.

- *crea_bus*: riflesso presente solamente in *run_da_shape.gaml* che si attiva durante il runtime ogni *frequenza_bus_standard* tick (frequenza settata all'inizio della simulazione e tenuta costante per tutta la durata di quest'ultima). Crea un bus in entrambi i capolinea di ogni linea ed imposta il lor senso di marcia (andata o ritorno)

3.4 L'algoritmo di ricerca dei percorsi possibili

Quando una persona viene generata essa possiede una fermata di partenza ed una fermata target, l'algoritmo ricorsivo *search_path* trova tutti i possibili percorsi che portano la persona alla sua fermata target entro un numero massimo di cambi *n_massimo_cambi*.

Nel corso della simulazione la fermata dove si trova momentaneamente la persona sceglierà dinamicamente quale di questi percorsi sia il più conveniente per la persona in oggetto applicando l'azione *scegli_percorso*.

L'azione *search_path* prende come input la fermata di partenza, la fermata target, il numero massimo di cambi e genera una lista di percorsi possibili che viene salvata all'interno della persona nella variabile *lista_cambi*. Ogni percorso possibile è una lista delle fermate a cui il passeggero deve cambiare, compresa la fermata target come ultimo elemento della lista.

Il limite superiore al numero di cambi effettuabili è stato inserito sia per rendere più realistica la simulazione, infatti si suppone che una persona non accetti di fare un percorso che gli richiede più di tre cambi, sia perchè in una rete connessa come quella utilizzata (dove tutti i nodi della rete sono raggiungibili da tutti gli altri con un determinato *path* finito) il costo computazionale del calcolo di tutti i percorsi possibili senza vincoli è molto elevato.

Sintassi dell'algoritmo:

```
action search_path(fermate start, fermate target, list
   <fermate> percorso, int iter, list<fermate> pool){
        ...
}
```

- **start:** Contiene la fermata di partenza della *persona* quando l'azione viene chiamata la prima volta. In tutte le chiamate ricorsive seguenti invece conterrà la fermata di cambio trovata nella chiamata precedente dell'azione.

- **target:** Contiene la fermata target della *persona* che ha chiamato l'azione. Viene passato come argomento alle successive chiamate ricorsive senza modifiche.
- **percorso:** Quando l'azione viene chiamata la prima volta dalla *persona* contiene semplicemente una lista vuota. Ad ogni iterazione dell'algoritmo verrà aggiunta una fermata in coda alla lista, tale fermata sarà un possibile cambio.
- **iter:** Rappresenta il numero massimo di cambi che è possibile effettuare, viene impostato inizialmente al valore desiderato alla prima chiamata dell'azione. Ad ogni iterazione viene diminuito di 1, quando raggiunge lo 0 significa che il *percorso* attuale non è più valido e deve essere scartato, in tal caso l'algoritmo viene interrotto e il *percorso* non viene salvato in *lista_cambi*.
- **pool:** Contiene la lista delle fermate possibili su cui vengono cercate le fermate di cambio, viene impostata inizialmente a tutte le fermate eccetto quella di partenza. Ad ogni iterazione vengono tolte da tale lista le fermate trovate come possibili cambi in modo da non riconsiderarle nelle iterazioni successive.

L'algoritmo utilizzato è ricorsivo e si sviluppa nella seguente maniera:

- Controlla se la condizione di successo è verificata, ovvero se la fermata *start* e la fermata *target* hanno linee in comune. In caso affermativo aggiunge la fermata *target* in coda alla lista di fermate *percorso* e aggiunge quest'ultima a *lista_cambi*, dopodiché termina.
- Controlla se la condizione di fallimento è verificata, ovvero se è stato raggiunto il numero massimo di iterazioni *iter* che corrispondono al numero massimo di cambi permessi. In caso affermativo termina senza salvare.
- L'algoritmo seleziona e salva in *new_start*, variabile interna, tutte le fermate appartenenti alle linee passanti per *start* per le quali passi almeno un'altra linea: queste fermate rappresentano tutti i possibili primi cambi per arrivare a *target*.
- L'algoritmo continua a verificare la presenza di linee comuni con il target come nel primo punto, per un massimo di *iter* volte, richiamando la funzione ricorsivamente e sostituendo *start* con ogni elemento di *new_start*.

3.5 La funzione utility

La funzione utility (azione *calcola_utility* della specie fermate) viene chiamata dinamicamente durante il runtime (nell'azione *scegli_percorso* della specie *persone*).

Ogni passeggero ha a disposizione diversi percorsi che lo portano alla propria destinazione (calcolati dall'**algoritmo di ricerca dei passaggi possibili** illustrato nel paragrafo precedente 3.4).

Ogni percorso è identificato da una lista di fermate, le fermate di cambio. Quando un passeggero si trova ad una fermata di cambio compie l'azione *scegli_percorso*, questa azione prende come input la lista dei percorsi possibili, ne calcola l'utility, sceglie quale percorso è il più vantaggioso ed elimina tutti i percorsi che non possono più essere presi una volta operata una scelta, ovvero elimina dalla lista dei percorsi possibili del passeggero tutti quelli che non hanno lo stesso primo cambio del percorso scelto.

L'azione *calcola_utility* tiene conto di diversi fattori:

- n_{cambi} : numero di cambi del percorso, abbiamo supposto che più è alto il numero di cambi da effettuare meno conveniente è la scelta di quel percorso.
- $n_{fermate}$: numero minimo di fermate che separa la fermata attuale dal cambio successivo, è importante osservare che è il numero MINIMO di fermate che separa le due fermate perchè lo stesso percorso può essere compiuto da linee diverse con un numero di fermate diverse, nel calcolo consideriamo la linea che fa meno fermate.
- ETA : tempo di arrivo stimato di un bus al cambio successivo, questo viene calcolato tramite una richiesta, da parte della fermata a cui si trova il passeggero, di compiere l'azione *calcola_ETA* a tutti i bus che passano per quella fermata e quella corrispondente al primo cambio.

L'*utility* di un possibile percorso viene dunque calcolata come:

$$u = \frac{1}{n_{cambi} + n_{fermate} + ETA}$$

La funzione *calcola_utility* sarà poi chiamata dall'azione *scegli_percorso* della persona, che controllerà ciclicamente tutte le utility dei percorsi possibili e sceglierà quello con il valore di utility più alto.

3.6 Setup del modello

Il setup del modello è implementato in *creapersone.gaml*. Lo scopo di questo file è di creare dei set di persone che possiedano ognuna una fermata di partenza, una fermata di arrivo ed un insieme di possibili cambi per arrivare all'obbiettivo e successivamente di salvare tali dati su diversi shapefile in modo da poter eseguire simulazioni a frequenze diverse con le stesse condizioni iniziali.

Il setup inizia con la creazione dell'ambiente virtuale presente dentro il *global init*. Le prime specie che vengono create sono le fermate, queste non sono altro che punti nello spazio con un valore di reference univoco, per ottenere le linee dei bus occorre raggrupparle in liste ordinate come spiegato nella sezione 3.1.

Dopodichè viene chiamata l'azione globale *accorpamento_fermate* descritta nella sezione 3.3, che crea una rete di fermate connessa (tutte le fermate possono essere raggiunte da tutte le altre in un numero finito di passi). Una volta chiamato il *global init* vengono eseguiti gli *init* delle singole specie.

Quando una fermata viene creata, a sua volta crea un numero fisso di persone (settato a 25) ognuna di queste ha come fermata di partenza la fermata che l'ha creata e come fermata target una assegnata a random tra le altre. In questo modo si ottiene una distribuzione uniforme di *fermate_target* sulla rete.

Quando una persona è creata viene inoltre calcolato l'elenco di tutti i possibili cambi che la persona può prendere per arrivare alla fermata di destinazione tramite l'algoritmo *search_path* descritto nella sezione 3.4. Se la persona non ha trovato nessun percorso per raggiungere il proprio target facendo un numero di cambi inferiore o uguale a *n_massimo_cambi*, viene distrutta.

Abbiamo fatto la scelta di non considerare le persone che non trovano un percorso con un numero di cambi inferiore o uguale a 3 perchè da analisi preliminari si è visto che la percentuale di tali persone, che andrebbero considerate avendo soddisfazione 0, è minima se si usa la rete completa dei trasporti e nulla se si usa la rete ridotta (comprendente solo le linee che coprono il centro cittadino), rete effettivamente usata per le nostre simulazioni.

Una volta create le persone con i relativi attributi, viene fatto uno shuffle sulla lista delle persone da salvare in modo da cercare di non avere più persone consecutive con la stessa fermata di partenza. In seguito le persone vengono suddivise in partizioni, ognuna delle quali viene salvata su uno shapefile diverso che conterrà un gruppo di persone con fermate di partenza eterogenee, la scelta è stata fare 60 partizioni. Questi shapefile vengono poi utilizzati in *run_da_shape* per immettere nuove persone nella rete.

3.7 Dinamica della simulazione

La dinamica della simulazione è implementata in *run_da_shape.gaml*. Sono stati realizzati due esperimenti: uno in **modalità GUI**, usato principalmente nelle fasi di sviluppo per verificare il corretto comportamento del modello, e uno in **modalità BATCH** nel quale è presente il codice per salvare su file i dati di output, usato per le simulazioni vere e proprie. La spiegazione su come utilizzare questi due esperimenti si trova nella sezione 4.

L'esperimento in batch mode consiste nell'effettuare un certo numero di simulazioni mantenendo fisse le condizioni iniziali, ovvero la distribuzione delle persone e dei loro target, e variando un singolo paramento: la frequenza di creazione dei bus ai rispettivi capolinea φ.

È stato deciso di dividere il modello in due file separati proprio per poter mantenere constanti le condizioni iniziali, scelta che inoltre ha permesso una maggiore velocità nell'esecuzione delle simulazioni.

Come spiegato in precedenza il primo file (*creapersone.gaml*) serve a creare un set di persone, ognuna con una fermata di partenza, una fermata di arrivo e un insieme di possibili cambi per arrivarci, informazioni che vengono poi salvare su diversi shapefile ed utilizzate nella fase di inizializzazione del secondo file (*run_da_shape.gaml*).

Nella fase di inizializzazione del modello viene creato l'ambiente virtuale in modo del tutto identico al file *creapersone.gaml* ed in più viene impostata la frequenza dei bus per la simulazione corrente.

Nel momento in cui la simulazione viene fatta partire entrano in gioco i riflessi globali che gestiscono la creazione degli agenti *persona* e *bus*.

Il riflesso *crea_persone* fa si che ad intervalli regolari di 120 secondi nuove persone siano create alle fermate. Ogni volta che il riflesso è eseguito viene selezionato uno degli shapefile generati da *creapersone.gaml* e vengono create tante persone quante sono presenti dentro quel file. Ad ogni chamata del riflesso, esso selezionerà lo shapefile successivo e così via, ricominciando poi dal primo in modo ciclico.

Si ricorda che essendo stato fatto uno shuffle sulle fermate di partenza delle persone in *creapersone.gaml*, le persone create dal riflesso *crea_persone* saranno distribuite in maniera omogenea all'interno della rete.

Il riflesso *crea_bus* crea ad intervalli regolari un bus per ogni capolinea di ogni linea; la durata dell'intervallo è determinato dalla frequenza φ impostata per la simulazione attuale.

Quello che avviene durante la simulazione è che ogni bus percorre la propria linea passando da una fermata all'altra scaricando e caricando passeggeri alle fermate in base alle loro destinazioni.

Il salvataggio dei dati di output si attiva dopo 4000 ticks in modo da prendere solo i dati "a regime", ovvero quando è passato un tempo sufficientemente lungo da avere una distribuzione dei bus sulla rete uniforme (inizialmente i bus si trovano tutti ai capolinea).

4 Utilizzo del programma

Questo paragrafo è dedicato all'utilizzo pratico del programma ai fini di svolgere una simulazione.

Per far funzionare il programma è necessario avere nella cartella *includes* del modello i file .csv estratti dal sito della GTT e lo shapefile *FermateGtt.shp* contenente i punti georeferenziati delle fermate. Inoltre è necessario creare manualmente una cartella *simulazioni* all'interno della cartella del progetto, dentro quest'ultima verranno salvati gli output delle simulazioni.

Ogni simulazione ha un suo *id* che deve essere impostato manualmente all'inizio di ognuna settando la variabile globale *id_simulazione* uguale in entrambi i file *creapersone.gaml* e *run_da_shape.gaml* (ad esempio per la prima simulazione si può impostare $id\_simulazione \leftarrow 0$)

La prima cosa da fare è creare gli shapefile contenenti le persone utilizzando *creapersone.gaml*, visto che gli *init* delle singole specie vengono eseguiti dopo

il *global init* non è possibile effettuare il salvataggio delle persone su shapefile all'interno del *global init* come sarebbe più logico pensare. Per questo motivo è stato creato un riflesso ad hoc chiamato *salva* che viene eseguito al primo tick e grazie al quale vengono salvate le persone in partizioni (come descritto nel paragrafo 3.6). Le partizioni sono di default 60, se si volesse modificarne il numero si deve cambiare il valore della variabile globale *n_partizioni* in entrambi i file.gaml. Inoltre viene anche creata una sottocartella *simN* di *simulazioni*, dove N è l'id della simulazione.

Dentro la cartella *simN* vengono a loro volta create due sottocartelle; *input* dove vengono salvati gli shapefile contenti i dati delle persone generati da *creapersone.gaml* e *output* che viene riempita da *run_da_shape.gaml*.

Una volta create le persone come descritto si può passare alla simulazione vera e propria che viene svolta dal modello in *run_da_shape.gaml*. Si ricorda che se non è già stato fatto in precedenza è necessario impostare la variabile *id_simulazione* allo stesso valore impostato in *creapersone.gaml*.

Esistono due possibilità di esperimento in *run_da_shape.gaml*, un esperimento GUI dove è possibile visualizzare i pullman che si muovono sulla rete con alcuni dati real-time, e un esperimento BATCH che permette di ricavare gli output analizzabili.

I dati real-time visualizzati sulla rete durante l'esperimento in modalità GUI sono le persone presenti ad ogni fermata.

Per far sì che ad ogni simulazione in BATCH MODE la frequenza sia diversa da quella della simulazione precedente viene utilizzato il file esterno *frequenza.txt* che contiene la lista delle frequenze già esplorate ed è aggiornato alla fine di ad ogni simulazione.

L'esperimento in BATCH MODE è stato implementato in modo che esegua di default 10 simulazioni diverse ognuna corrispondente ad una frequenza diversa. Ognuna di queste è composta da due fasi, la prima è l'esecuzione della simulazione vera e propria e la seconda è l'escuzione della azione *_step_*.

Per eseguire l'esperimento in BATCH MODE occorre impostare il valore di *repeat* a $N+1$ e il valore di *repeat_meno_uno* a N, dove N è il numero di simulazioni che si vogliono svolgere, è necessario aggiungerne una in più perchè la prima fase della prima simulazione viene saltata. Infatti è presente un controllo all'inizio del *global init* che verifica l'esistenza del file *frequenza.txt* e in caso negativo la simulazione viene abortita e si passa alla seconda fase dove viene creato il file.

E' stato necessario appoggiarsi ad un file esterno *frequenza.txt* in quanto non è possibile modificare i parametri della simulazione prima che questa venga eseguita e perché durante la simulazione non è possibile accedere alle variabili di *_step_*. Dunque appoggiarsi ad un file esterno è l'unico modo che è stato trovato per avere frequenze diverse ad ogni simulazione.

Per utilizzare l'esperimento in modalità GUI è necessario creare manualmente il file *frequenza.txt* dentro la cartella *output* della simulazione contenente il valore di frequenza desiderato in ticks.

5 Descrizione degli output del modello

Gli output analizzati nella sezione 6 sono prodotti dal modello in *run_da_shape.gaml* e vengono salvati nella cartella *output* di ogni simulazione.

I file salvati sono di due tipi, **efficienzaBus_alla_freq_f.csv** e **insoddisfazione_alla_freq_f.csv** dove f è la frequenza utilizzata in quello step (il range esplorato è 300 ticks $\leq f \leq$ 1380 ticks) che corrisponde al range di frequenza tra 5 e 23 minuti.

Ad ogni riga del file *efficienzaBus_alla_freq_f.csv* corrisponde un viaggio di un pullman, la prima colonna contiene la linea a cui appartiene il bus, la seconda colonna contiene l'efficienza del viaggio di quel bus, la terza colonna contiene i passeggeri trasportati dal bus e la quarta e ultima colonna contiene la capacità massima del bus.

Ad ogni riga del file *insoddisfazione_alla_freq_f.csv* corrisponde il viaggio di una persona, la prima colonna contiene l'id della persona, la seconda colonna contiene la distanza percorsa da quella persona in linea d'aria, la terza colonna contiene il tempo impiegato per giungere a destinazione e la quarta e ultima colonna contiene l'insoddisfazione della persona.

6 Analisi dei dati e risultati

L'analisi dei dati raccolti è stata fatta in python utilizzando l'*Ipython notebook* allegato **Analisi_DEFINITIVE.ipynb** di cui è presente anche una copia in html.

Per ogni frequenza f compresa tra 300 ticks $\leq f \leq$ 1380 ticks a passi di 120 ticks sono state fatte 51 simulazioni con condizioni iniziali diverse (diversa distribuzione delle persone nella rete).

Dato che le misure dell'efficienza e della soddisfazione ad ogni frequenza sono misure ripetute che generano valori indipendenti si è assunto che la distribuzione delle misure attorno al valore vero sia gaussiana, e quindi sono state usati come indicatori statistici la media e la deviazione standard.

Fissata una simulazione i con $0 \leq i < 51$ è stata calcolata la soddisfazione media $\bar{s}_i^f$ dei passeggeri per ogni frequenza f esplorata come:

$$\bar{s}_i^f = \frac{1}{N} \sum_{n=1}^{N} (s_i^f)_n$$

dove N è il numero totale di persone che ha concluso il suo viaggio nella simulazione i-esima, questo è stato fatto utilizzando la funzione *numpy.mean*().

Il relativo errore è stato calcolato come la deviazione standard ridotta utilizzando la funzione *numpy.std*() di python:

$$\sigma_i = \sqrt{\frac{\sum_{n=1}^{N} \left((s_i^f)_n - \bar{s}_i^f \right)^2}{N-1}}$$

Dopodichè è stata calcolata la soddisfazione totale s_{tot}^f del cliente alla frequenza f come:

$$s_{tot}^f = \frac{1}{51} \Sigma_{i=1}^{51} \bar{s}_i^f$$

e il relativo errore è stato calcolato usando la propagazione dell'errore:

$$\sigma_{tot} = \frac{1}{51} \sqrt{\sum_{i=1}^{51} \sigma_i^2}$$

Lo stesso processo è stato fatto per calcolare l'efficienza totale e il relativo errore, i risultati di tali calcoli sono riassunti nella tabella 1 e sono visualizzati nella figura 4.

Si osserva che all'aumentare della frequenza l'errore sulla soddisfazione diminuisce, mentre quello sull'efficienza aumenta. Tuttavia osservando la tabella 1 si nota un aumento di entrambi gli errori relativi all'aumentare della frequenza per entrambe le grandezze. Questo comportanto è dovuto ad una minore quantità di dati statistici, infatti nello stesso arco di tempo, all'aumentare della frequenza vengono creati meno bus nella rete con conseguente diminuzione di persone che

raggiungono il target.

Una volta ricavati gli errori sulle misure dell'efficienza e della soddisfazione ad ogni frequenza, ai fini di trovare l'intersezione delle due curve e la frequenza ottimizzata φ^* della rete, si è cercata la forma funzionale più adatta ad interpolare i dati relativi alle due curve nel range di frequenze esplorate.

Sono state fatte le medesime ipotesi nulle sull'andamento dei dati per le curve di efficienza e soddisfazione: lineare, quadratica e cubica, i grafici delle interpolazioni sono visualizzabili nelle figure 5 e 6.

I dati sono stati interpolati usando la funzione *numpy.polyfit* di python:

numpy.polyfit(x, y, deg, rcond=None, full=False, w=None, cov=True)

funzione che prende come input i punti (x, y), da interpolare con un polinomio di grado *deg* ed i pesi $w = \frac{1}{\sigma}$ dove σ è l'errore sui punti e restituisce i coefficienti che minimizzano:

$$E = \sum_{j=0}^{k} |w_j * (p(x_j) - y_j)|^2$$

dove $p(x_j)$ sono i valori del polinomio calcolato in x_j.

La funzione restituisce inoltre, settando il booleano $cov = True$, la matrice delle covarianze. Maggiori informazioni sull'implementazione dell'algoritmo per il calcolo dei coefficienti si possono trovare sulla documentazione ufficiale numpy.polyfit().

Per determinare quali delle diverse ipotesi nulle fosse quella più corretta è stato dunque fatto un **test del** χ^2 i cui risultati sono riassunti nella tabella 2, calcolando

$$\chi^2 = \sum_{i=1}^{N} \frac{(y_i - f(x_i))^2}{\sigma_i^2}$$

con $D.F. = N - m$ dove N sono la numerosità dei dati e m sono i parametri stimati (per la lineare $D.F. = 8$, per la parabolica $D.F. = 7$, per la cubica $D.F. = 6$).

Si osserva che, supponendo un livello di significatività $\alpha = 0.05$, la migliore ipotesi nulla relativa all'interpolazione dei dati della soddisfazione è l'ipotesi *lineare*.

Si osserva inoltre che, sempre assumendo il livelli di significatività $\alpha = 0.05$, i valori di chi quadro per le interpolazioni della curva delle efficienze sono molto piccoli. Normalmente questo è imputabile ad una sovrastima degli errori o ad una interpolazione con una funzione con troppi gradi di libertà. Per questo motivo si è deciso di testare l'ipotesi nulla *costante* per l'interpolazione dell'efficienza, osservando tuttavia un valore di χ^2 troppo alto per poterla accettare. Si è dunque scelta come *lineare* la curva che meglio interpola i dati relativi all'efficienza del servizio.

Infine è stata cercata l'intersezione tra le due curve che corrisponde alla frequenza ottimizzata φ^* della rete, e si è calcolato il relativo errore tramite propagazione degli errori sui parametri stimati delle interpolazioni mostrati in tabella 3.

Il risultato ottenuto per la la frequenza ottimizzata della rete è:

$$\varphi^* = (16.95 \pm 1.17) \text{ minuti}$$

Questa frequenza permette di ottere un'efficienza eff^* e una soddisfazione $sodd^*$ ottimizzate pari a:

$$eff^* = sodd^* = 0.19 \pm 0.02$$

Si osserva che si è raggiunto solo il 34% della soddisfazione e il 22% dell'efficienza massime possibili nei due estremi del range di frequenze esplorate.

Inoltre la frequenza ottimizzata permette di aumentare dell'88% l'efficienza del servizio pagando una diminuzione del 55% della soddisfazione del passeggero rispetto ai valori che massimizzano la soddisfazione nel range di frequenze esplorate.

I valori trovati non hanno pretesa di essere applicabili alla rete reale della GTT, in quanto si è considerata una sola frequenza per tutti i bus, la rete utilizzata per le simulazioni non è completa di tutte le linee e la distribuzione delle *fermate_target* e delle *fermate_partenza* non tiene conto della reale distribuzione della popolazione sul territorio.

Le implementazioni future del progetto potrebbero tendere proprio a migliorare questi aspetti.

7 Tabelle e grafici

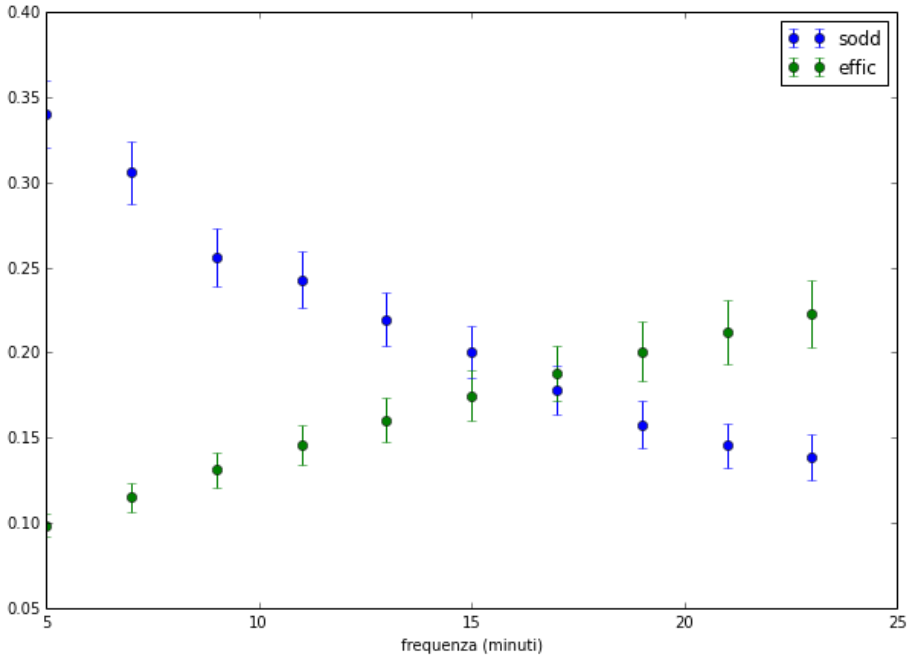

Figura 4: Andamento della soddisfazione e dell'efficienza in funzione della frequenza

frequenza (minuti)	**soddisfazione**	σ_s	σ_s **relativo**	**efficienza**	σ_{eff}	σ_{eff} **relativo**
5	0.340	0.020	0.058	0.098	0.007	0.068
7	0.306	0.019	0.061	0.115	0.009	0.077
9	0.256	0.017	0.067	0.131	0.010	0.078
11	0.243	0.017	0.069	0.146	0.012	0.080
13	0.220	0.016	0.072	0.160	0.013	0.082
15	0.200	0.015	0.077	0.175	0.015	0.085
17	0.178	0.015	0.081	0.188	0.016	0.086
19	0.158	0.014	0.088	0.201	0.018	0.088
21	0.145	0.013	0.092	0.212	0.019	0.088
23	0.138	0.013	0.096	0.223	0.020	0.089

Tabella 1: Valori della soddisfazione e dell'efficienza alle diverse frequenze con errori assoluti e relativi

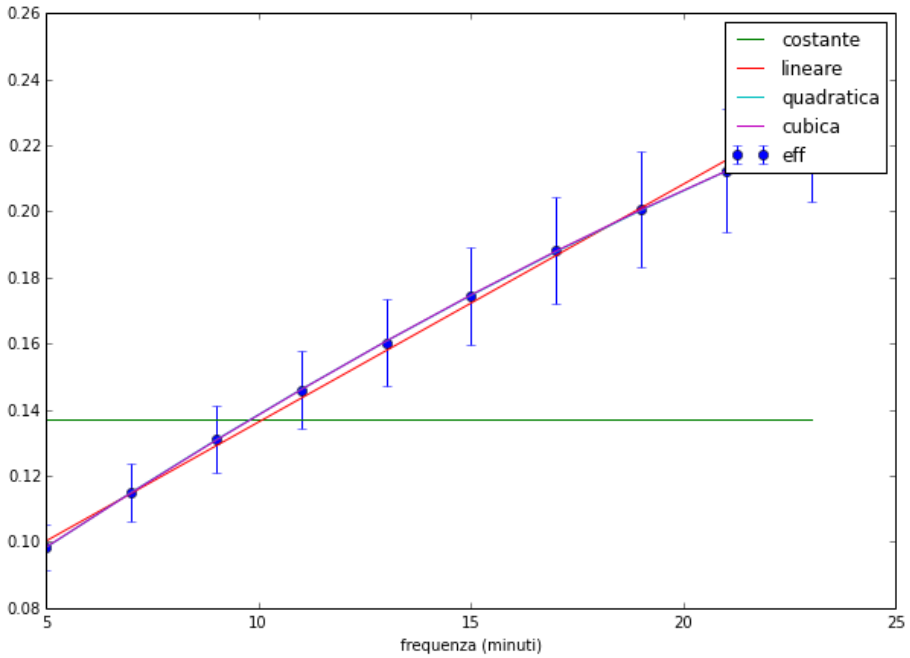

Figura 5: Interpolazione dei dati dell'efficienza con diverse ipotesi nulle

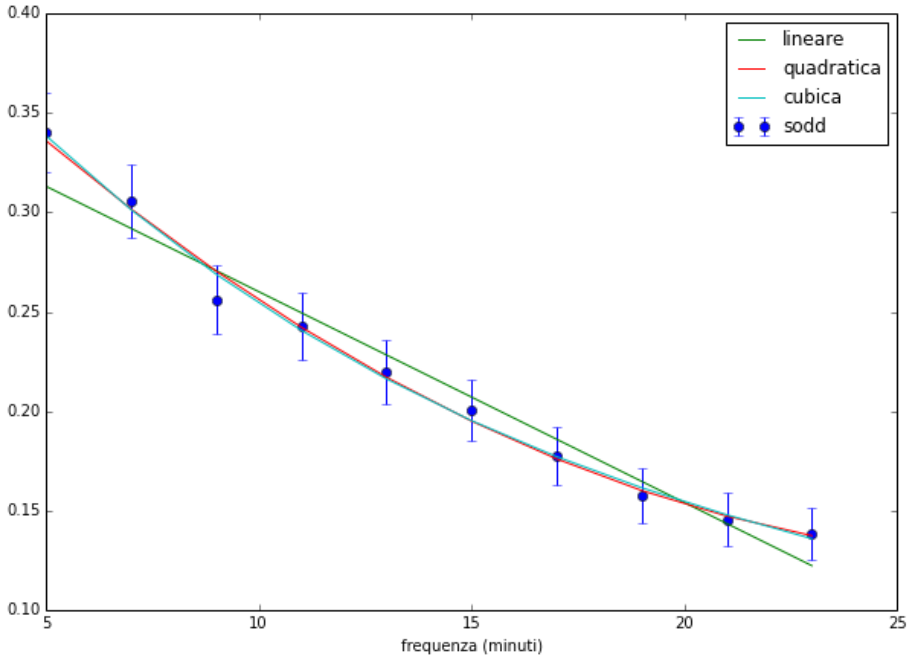

Figura 6: Interpolazione dei dati della soddisfazione con diverse ipotesi nulle

ipotesi	**soddisfazione** χ^2	**efficienza** χ^2
costante	-	107.905
lineare	5.890	0.389
quadratica	1.029	0.003
cubica	0.960	0.002

Tabella 2: Test del chi quadro per diverse ipotesi nulle

ipotesi lineare $y = ax + b$

	a	σ_a	b	σ_b
efficienza	0.0069	0.0002	0.068	0.003
soddisfazione	-0.011	0.001	0.373	0.014

Tabella 3: Parametri stimati con errori