

UNIVERSITÀ DEGLI STUDI  
DI MODENA E REGGIO EMILIA

DIPARTIMENTO DI FISICA, INFORMATICA E  
MATEMATICA  
CORSO DI LAUREA IN INFORMATICA

**CREAZIONE ED ANALISI DI  
SCENARI DI MOBILITÀ URBANA  
ATTRAVERSO IL SIMULATORE  
OPEN SOURCE SUMO**

**Salvatore Bianco**

Tesi di Laurea

*Relatore:*

*Claudia Canali*

*Correlatore:*

*Riccardo Lancellotti*

Anno Accademico 2021/2022



## Ringraziamenti

## Indice

.....	1
.....	1
Introduzione.....	10
Internet del futuro.....	12
1.1 Smart City.....	12
1.1.1 Mobilità Urbana Smart.....	13
1.2 Architettura di una Smart city.....	15
1.2.1 Tecnologie.....	16
1.3 Cloud Computing.....	17
1.3.1 Tipi di Cloud computing.....	21
1.3.2 Vantaggi e limiti del cloud computing.....	22
Edge/Fog Computing.....	24
2.1 Introduzione edge/fog computing.....	24
2.1.1 Suddivisione su più livelli.....	25
2.2 Fog computing.....	27
2.2.1 Caratteristiche del Fog Computing.....	27
2.2.2 Applicazioni del Fog computing con Smart City.....	32
2.2.3 Vantaggi e svantaggi del Fog computing.....	32
2.3 Edge Computing.....	33
2.3.1 Differenze e somiglianze con il Fog Computing.....	34
2.3.2 Esempi di scenari applicativi di edge computing e dispositivi diffusi.....	35
2.3.3 Evoluzione dell' Edge computing.....	37
2.3.4 Vantaggi e svantaggi dell'edge computing.....	37
SUMO.....	39
3.1 Descrizione del Problema.....	39
3.1.1 Obiettivi, tecnologie utilizzate e ambiente di lavoro.....	40
3.2 Simulation of Urban MObility.....	40
3.2.1 Creazione di reti.....	41
3.2.2 Entità della rete.....	44
3.2.3 Configurazione.....	48
3.3 Tools.....	52
3.3.1 osmWebWizard.....	52
3.3.2 randomTrips.....	54
3.3.3 Altri tool.....	56

3.4 Output.....	57
3.4.1 tools aggiuntivi di analisi output.....	59
3.4.2 Analisi dei tool su scenario di prova.....	61
3.5 Simulazione completa di uno scenario reale.....	64
3.5.1 Creazione net-file.....	64
3.5.2 Creazione del file route.....	65
3.5.4 Modifica file di configurazione e simulazione.....	66
3.5.5 Output e analisi dei dati ottenuti.....	72
Conclusioni.....	76
Bibliografia.....	77

## Indice delle figure

Figura 1: Rappresentazione generale urban smart mobility .....	20
Figura 2: Rappresentazione generale infrastruttura smart city .....	23
Figura 3: Rappresentazione cloud computing .....	25
Figura 4: Architettura cloud computing .....	26
Figura 5: Rappresentazione layer cloud, fog/edge computing .....	30
Figura 6: Fog visto come estensione del cloud ma più vicino ai dispositivi finali .....	33
Figura 7: Fog computing e applicazioni IoT .....	36
Figura 8: Rappresentazione della divisione a livelli edge/fog in una città intelligente. ....	40
Figura 9: Netedit interfaccia grafica.....	46
Figura 10: Input per creazione rete tramite Netconvert .....	47
Figura 11: Esecuzione di netconvert da linea di comando.....	48
Figura 12: Definizione di un'entità nel file.rou.xml.....	49
Figura 13: Definizione dell'entità persona in file.rou.xml.....	50
Figura 14: Esempio di definizione multipla dell' entità persona in file.rou.xml.....	51
Figura 15: Duarouter eseguito da riga di comando.....	52
Figura 16: Contenuto di un file.sumocfg.....	54
Figura 17: Esempio di file.sumocfg più dettagliato.....	54
Figura 18: Sinossi sumo.....	55
Figura 19: Impostazione temporanea della variabile d'ambiente.....	57
Figura 20: browser web aperto da osmWebWizard.py.....	58
Figura 21: File ottenuti dallo scenario generato tramite osmWebWizard.py.....	58
Figura 22: Esempio di un nuovo tipo di veicolo .....	60
Figura 23: file trips.trips.xml.....	60
Figura 24: Output di plotXMLAttribute.py.....	61
Figura 25: Esempio definizione output in file di configurazione.....	62
Figura 26: Rappresentazione in formato xml di un output di floating car data (fcd-output).....	63
Figura 27: Immagine generata da xmlAnalyzer_graphic.py.....	65
Figura 28: Rete di prova visibile nella GUI di netedit.....	66
Figura 29: chiamata di xmlAnalyzer.py.....	66
Figura 30: Rappresentazione output di xmlAnalyzer.py.....	67
Figura 31: Chiamata del tool xmlGridstats.py.....	67

Figura 32: Rappresentazione output xmlGridstats.py.....	67
Figura 33: Contenuto del file osm.net.xml visualizzato su netedit.....	69
Figura 34: Elemento input del fileosm.sumocfg.....	71
Figura 35: Elemento output nel file di configurazione.....	71
Figura 36: Frammento dell'output generato.....	71
Figura 37: Frammento dell'output di xmlgridStats.py.....	72
Figura 38: Frammento dell'output generato da xmlAnalyzer.py.....	73
Figura 39: Analisi dei dati di xmlAnalyzer.py.....	80
Figura 40: Analisi dell'output di xmlGridStats.py.....	81

# Introduzione

La costante evoluzione di Internet ha portato dei cambiamenti in ogni campo. Ad oggi è sempre più al centro delle nostre vite, e si cerca di trovare sempre un nuovo campo nel quale introdurlo per migliorarlo.

Questa continua evoluzione di Internet e, conseguenzialmente, anche delle tecnologie che ne permettono l'utilizzo si è arrivati a parlare di IoT, oggetti con una propria identità digitale che fino a dieci anni fa nessuno avrebbe mai pensato di utilizzare come vengono utilizzati oggi, ma che oggi compongono alcuni dei progetti più ambiziosi del paradigma IoT come ad esempio le Smart City.

Per permettere sviluppo di applicazioni nell'ambito IoT , ma nello specifico delle Smart City è necessario il paradigma del Cloud Computing. Infatti grazie a quest'ultimo è possibile ad oggi permettere la distribuzione di servizi software e hardware da remoto. Alcuni esempi sono i sistemi di archiviazione da remoto che sono diffusissimi oggi.

Grazie alla diffusione del paradigma cloud è stato possibile iniziare qualche test sulla realizzazione di città intelligenti. D'altro canto con l'aumentare della tecnologia sono aumentati anche i dati da dover gestire e analizzare in tempi brevi, quindi sono nati i primi problemi legati alla latenza, che ad oggi è uno dei principali, se non il principale , problema per la realizzazione di auto a guida autonoma, progetto che richiede tempi di analisi dei dati praticamente al decimo di secondo. Per far fronte a questo problema vengono introdotti due nuovi paradigmi che non vanno a sostituire il paradigma cloud ma lo vanno a sostenere, e sono Edge computing e Fog computing. Con la loro implementazione l'architettura dei servizi cloud cambia da un singolo livello a un'architettura piramidale, in grado di partire dal livello più in basso (edge) dove i dati vengono generati e arrivare , dopo una serie di pre-elaborazioni, al cloud centrale passando dai nodi fog, riducendo così moltissimo i tempi di attesa. Inoltre nello sviluppo di città intelligenti, dove i dati sono generati proprio ai nodi edge, da sensori o telecamere di sicurezza questi paradigmi sono fondamentali, basti pensare ad esempio a scenari di guida autonoma nei quali i dati vengono elaborati con ritardo e così si rischierebbero incidenti perché non è stato rilevato in tempo un pedone.



Da questo punto di vista, inoltre, un ruolo fondamentale è svolto dalle simulazioni che sono l'imitazione del funzionamento di un reale scenario. Tramite la simulazione si possono valutare e prevedere una serie di eventi, come il comportamento dei pedoni, utile per la mobilità urbana.

L'obiettivo di questa tesi è proprio quello di interfacciarsi con un simulatore, studiarlo e sviluppare tracce reali sulle quali svolgere le simulazioni per andare a generare dati realistici. Quindi ci sarà un'ampia panoramica su come costruire una simulazione correttamente, sia tramite interfaccia grafica che da riga di comando. I dati ottenuti saranno analizzati tramite gli strumenti messi a disposizione dal simulatore.

Nello specifico l'elaborato è così composto : capitolo primo, introduzione su internet e poi ampia panoramica su smart city e sulla sua architettura, approfondendo il cloud computing. Nel secondo capitolo , vengono presentati i paradigmi edge e fog, approfonditi fino a vederne anche i limiti. Nel terzo, ed ultimo capitolo, sarà presentato ampiamente il simulatore Sumo, vedendo ogni aspetto che permetta di fare una simulazione realistica e un'analisi realistica.

# Capitolo 1

## Internet del futuro

In questo capitolo si parla di Internet moderno e di come oggi viene sfruttato. Nello specifico si illustrano i miglioramenti che l'IoT ha portato nella vita delle persone, guardando nello specifico il mondo delle Smart City e la smart mobility, che proprio grazie all'unione di Internet con gli oggetti con una propria identità digitale che ad oggi permette di gestire alcuni aspetti cittadini come il controllo dei rifiuti, ma che un giorno futuro vedrà realizzarsi progetti molto ambiziosi come la guida autonoma. Questi oggetti generano e raccolgono molti dati per questo, il tempo di gestione e filtraggio di questi dati è molto importante per la realizzazione delle Smart City ma anche per il monitoraggio in tempo reale della mobilità, che è alla base anche per il progetto di guida autonoma.

### 1.1 Smart City

Città intelligente, nient'altro che un nuovo modo di progettazione urbana con il fine di migliorare la vita quotidiana in modo sostenibile. Il concetto di smart city diventa reale nel 2009 a Rio de Janeiro con il piano di innovazione e per la gestione dei rifiuti con il fine di migliorare la vita urbana. Solo 5 anni dopo vengono a New York chiariti i punti essenziali per definire una città smart, che seppur differenti tra loro mirano tutti a migliorare la vita dei cittadini. Si stima che nel 2050 la maggior parte della popolazione sarà concentrata in città quindi è necessario pensare ed attuare delle modifiche per la gestione dei rifiuti, delle risorse, della sanità e del tempo. Come esempio, per comprendere meglio come una città intelligente possa migliorare la qualità della vita, basterebbe pensare che secondo l'OMS ogni anno circa 7 milioni persone nel mondo hanno problemi di salute collegabili all'inquinamento atmosferico, il che potrebbe essere risolto con una mobilità intelligente nelle città, con ad esempio aree pedonali e maggiori servizi di green-sharing oppure un trasporto urbano green efficiente. Per realizzare queste modifiche le tecnologie svolgono un ruolo importante. Dai semafori intelligenti alle videocamere di sorveglianza, tutti in comunicazione tra loro tramite infrastrutture solide, possiamo dire che i sensori sono gli attori finali senza i quali difficilmente si potrebbe parlare di smart city. L'obiettivo finale, quindi, è quello di fornire servizi che riguardano salute, scuola, e mobilità.

### **1.1.1 Mobilità Urbana Smart**

Con i continui allarmi lanciati sul cambiamento climatico e l'inquinamento, tutti gli interessi e le strategie trovano un punto comune, quello del miglioramento della mobilità urbana. Ad oggi il trasporto urbano e il trasporto di merci sono i fattori principali dell'inquinamento, per quanto riguarda l'Europa circa il 20% di CO<sub>2</sub>, e quindi hanno anche un peso sulla salute delle persone. Sono molteplici le richieste, soprattutto dalle autorità governative per una svolta nel trasporto verso un cambiamento "Smart" andando a risolvere i problemi generali, perché la mobilità urbana, e quindi trasporto di merci e persone, in particolare quest'ultimo che ingloba il trasporto singolo e collettivo che sono due tipi di trasporti gestiti separatamente, il primo è frutto delle scelte dell'individuo che decide di muoversi con il suo veicolo o senza (auto, moto, bicicletta o a piedi); il secondo è frutto delle scelte dei gestori del servizio. Indipendentemente dal tipo di trasporto però la strada è l'unica alternativa per spostarsi andando quindi a creare problemi al traffico, sull'ambiente e sulla sicurezza. Per ovviare a questo problema la tecnologia svolge un ruolo centrale portando miglioramenti alla sostenibilità grazie ad una gestione dei sistemi in modo efficiente, riducendo anche il consumo di energia. La tecnologia è applicabile sia ai mezzi di trasporto che alle persone. Un esempio che ci accompagna ormai da molti anni è il GPS i cui dati possono segnalare eventuali congestioni del traffico o semplicemente utilizzato per controllare i tempi di percorrenza. Ovviamente l'unione tra la mobilità urbana e la tecnologia non riguarda solo GPS, altri scenari prevedono il collegamento tra veicoli (V2V) o tra veicoli e infrastrutture (V2I) o anche veicoli con tutto (V2X), ad esempio veicoli collegati ai semafori che in base allo stato di quest'ultimo variano la loro velocità andando ad impattare in maniera non indifferente sui flussi di traffico. Quindi veicoli che non solo sapranno calcolare percorsi migliori ma che sapranno gestire in tempo reale qualsiasi tipo di situazione, una base che permetterà l'arrivo della guida autonoma. Questi cambiamenti porteranno a quello che è definito come green mobility, in cui rientra anche la mobilità attiva, maggiori piste ciclabili e maggiori aree pedonali. I pedoni o ciclisti a loro volta devono far parte dei dati esaminati dalle infrastrutture tecnologiche perché rientrano nei casi di interazione con i veicoli, basti pensare agli attraversamenti pedonali vicino delle aree pedonali dove la probabilità di incidenti con dei veicoli è maggiore, e questo è uno dei

fattori che rientra nell'analisi dei rischi di incidenti pedone/veicolo. Nello specifico per questo tipo di analisi si devono esaminare i seguenti fattori:

- Probabilità che l'attraversamento pedonale possa dar luogo ad una interazione con i veicoli ;
- La gravità del danno frutto dell'interazione veicolo/pedone

Le interazioni veicolo/pedone sono classificate in relazione alla loro entità e alla possibilità di una collisione, solo dopo si procederà con la valutazione della probabilità di interazione della frequenza. Quindi l'uso della tecnologia per la mobilità urbana non è fondamentale solo per dal punto di vista del traffico e dei veicoli, andando così a migliorare il clima , l'aria e quindi anche impattando sulla salute, ma è ottimo anche per la sicurezza dei cittadini grazie a questi tracciamenti.

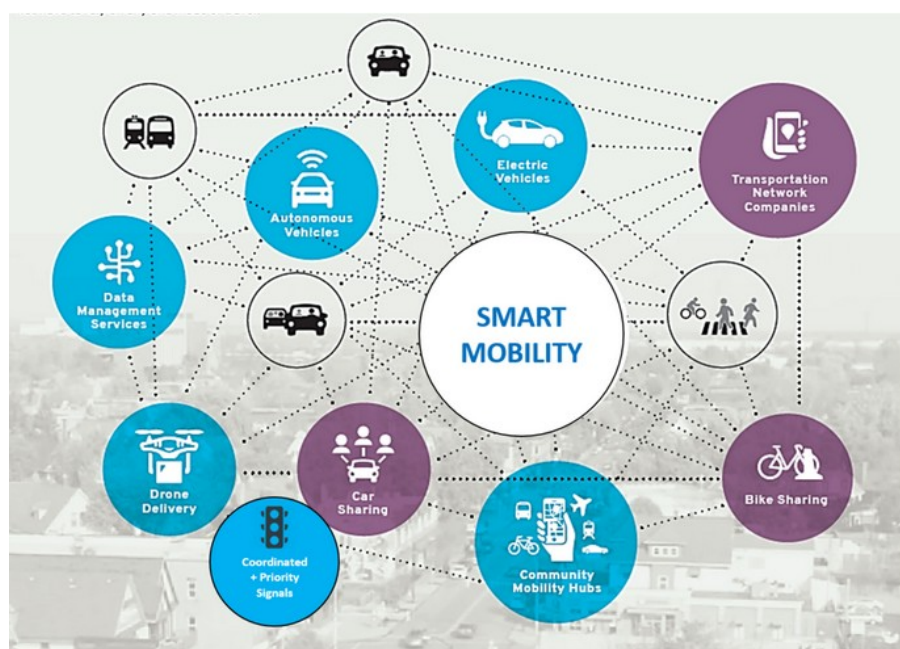


Figura 1: Rappresentazione generale urban smart mobility<sup>1</sup>

<sup>1</sup> Risorsa: <https://www.gbnrtc.org/smartmobility>

## 1.2 Architettura di una Smart city

Dopo aver parlato del fattore più importante , ovvero l'innalzamento della popolazione nelle città entro il 2050 e quindi della gestione di differenti problematiche che si andranno a creare (mobilità ,salute ,sicurezza ,etc),bisogna ora analizzare come una smart city è strutturata architettturalmente e come vengono gestiti componenti come trasporti, salute casa ed energia.

I dati generati da queste infrastrutture provengono per la maggior parte da reti di sensori wireless, ma anche telecamere e altri dispositivi che rientrano nel mondo IoT.

Come detto in precedenza le soluzioni innovative per le città sono indispensabili per affrontare gli effetti sociali, economici, e ambientali e questo implica che una città intelligente , per essere definita tale, che fornisca una migliore efficienza urbana attraverso la combinazione tra tecnologia e infrastrutture, tra tecnologia e architettura ma anche con gli oggetti quotidiani o noi stessi, tecnologie definite come TIC , tecnologie dell'informazione e della comunicazione, Quindi si comprende come tutto lo sviluppo di una Smart City ruoti intorno alle TIC.

Per una corretta progettazione e implementazione di città smart è necessario avere esperti di più campi , tra cui TIC, ingegneria, politica, economia. Le basi della struttura di una città intelligente sono :

- Infrastruttura a banda larga: molto importante ,sia cablata che wireless, anche se tutt'oggi è difficile fornire una banda larga wireless che risulterebbe fondamentale vista la crescita esplosiva di applicazione mobile, diffusione e connettività di tutti i dispositivi smart, perché è fondamentale che i cittadini siano costantemente connessi per condividere conoscenze ed esperienze;
- Servizi elettronici: ovviamente necessitano delle TIC per fornire servizi di vendita o consegna.

Ovviamente anche l'aspetto politico non è da sottovalutare ,infatti da ambienti governativi provengono richieste che cercano di puntare l'attenzione sulla sostenibilità , richiedendo che le infrastrutture siano sostenibili, quindi di aumentare l'efficienza di una città a 360° comprendendo anche le infrastrutture energetiche.

### 1.2.1 Tecnologie

Dopo aver visto come è strutturata una smart city e alcuni aspetti fondamentali per essere definita tale , è d'obbligo anche analizzarla dal punto di vista tecnologico e della gestione dei dati, grazie al quale il numero di iniziative di città intelligenti è aumentato. Questi progressi sono il frutto dell'ampia diffusione di dispositivi mobili che consentono al cittadino di essere parte attiva degli ambienti urbani e metropolitani, definita come componente urbana. Questo importante ruolo ha un po' cambiato anche il punto di vista sulle tecnologie utili ora a rendere le città intelligenti ma soprattutto a migliorare la vita quotidiana dei cittadini.

Innanzitutto gli elementi tecnologici si dividono tra hardware :

- sensori
- apparecchiature wireless
- telecamere

Ma ovviamente anche grazie al connubio con i software necessari per formare quegli ambienti intelligenti:

- intelligenza artificiale
- sistemi esperti

Dai singoli componenti ICT possiamo quindi arrivare alla creazione di smart city attraverso la loro unione , combinare alle tecnologie e alle infrastrutture software intelligenti . Come già è stato detto , tra le applicazioni più usate e sviluppate troviamo i GPS per i trasporti , database ora vengono usati per monitorare e registra stati di salute, oppure uso della tecnologia per migliorare istruzione e efficienza energetica. Le città intelligenti ovviamente necessitano anche di un'analisi rapida e una gestione sicura dei dati , che non è possibile realizzare attraverso i classici database perché non in grado di elaborare una quantità così grande di informazioni; l'implementazione di migliaia di sensori e dispositivi che una città intelligente pone sfide significative per la gestione e l'elaborazione dei questi big data. Copiare tutto da ogni sistema non è pratico , è necessaria un'infrastruttura che sia scalabile nell'archiviazione, gestione ed elaborazione, infrastruttura che oggi è conosciuta come

cloud-computing, che consente l'accesso alle risorse in modo condiviso ed è affidabile, permettendo così l'accesso a determinati servizi qualora le persone lo richiedessero.



*Figura 2: Rappresentazione generale infrastruttura smart city*

Fonte: [https://www.researchgate.net/figure/A-smart-city-architecture-abstract-view-From-the-technical-point-of-view-the\\_fig1\\_326874508](https://www.researchgate.net/figure/A-smart-city-architecture-abstract-view-From-the-technical-point-of-view-the_fig1_326874508)

## 1.3 Cloud Computing

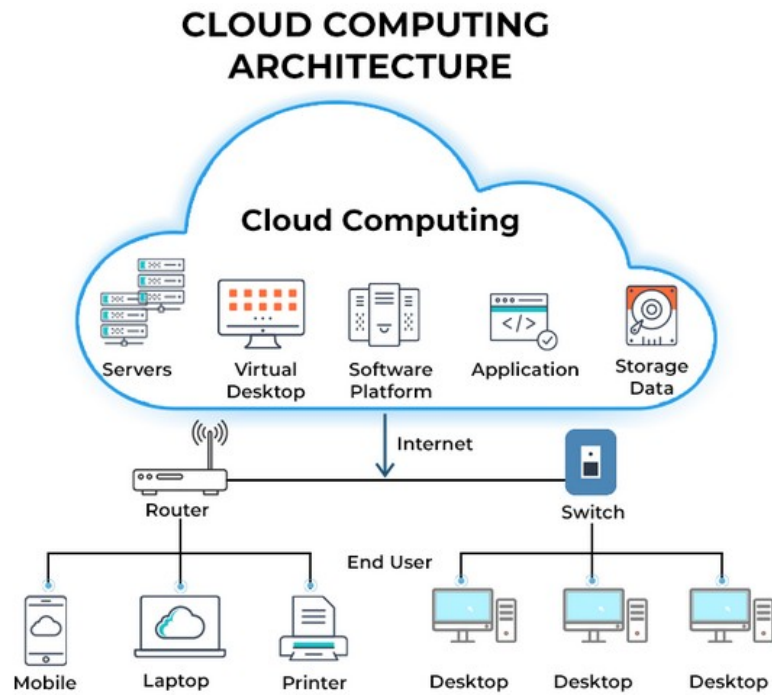
Come visto nei precedenti paragrafi, viviamo in un'epoca in cui vengono generati grandi quantitativi di dati che richiedono un approccio differente per la gestione e l'elaborazione. La tecnologia che maggiormente viene affiancata a questi dati è la tecnologia cloud. Il cloud computing, come comprensibile dal nome offre servizi di computing come software, archiviazione, elaborazione dati, tutto ovviamente tramite una connessione internet. Questi servizi sono erogati da aziende cloud provider, che forniscono questi servizi su richiesta e facendo pagare solo i servizi utilizzati. Quindi i servizi non risiedono nei server dell'azienda, ma risiedono su macchine remote, in questo modo agevolano i server locali aziendali per la gestione di altri tipi di servizi utili all'azienda. Molte aziende negli anni si sono avvicinate al cloud computing poiché hanno notato che avrebbero ottenuto più benefici :

- Spese ridotte per manutenzione ;
- Accesso da remoto ai servizi;

- Alta scalabilità.

*Figura 3: Rappresentazione cloud computing*

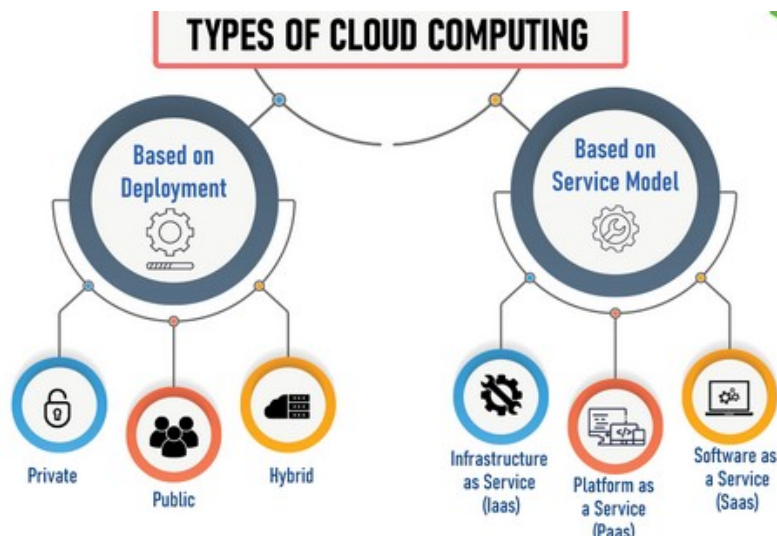
*Fonte: <https://www.spiceworks.com/tech/cloud/articles/what-is-cloud-computing/>*





### 1.3.1 Tipi di Cloud computing

Il Cloud computing, come detto nel paragrafo precedente, non offre un singolo servizio ed è diviso in base al tipo di cloud.



*Figura 4: Architettura cloud computing*

Fonte: <https://www.spiceworks.com/tech/cloud/articles/what-is-cloudcomputing/>

Come visibile nella figura 4 , il cloud computing si divide sia in tipologia che in base ai servizi offerti. Con la divisione in base al tipo, il cloud è classificabile come cloud privato, pubblico e ibrido.

- Cloud privato: servizi offerti su una rete priva, esclusivi per una singola organizzazione. Un esempio è un cloud aziendale creato per le necessità aziendali e non accessibile da attori esterni all'organizzazione. I vantaggi sono gli stessi di un cloud pubblico. Dal punto di vista della sicurezza il livello è elevato per garantire che i dati non siano accessibili dall'esterno. Gli svantaggi di un cloud privato sono tutti sulla gestione , che ricade sull'azienda;
- Cloud pubblico: come intuibile dal nome, offrono servizi accessibili tramite internet a tutti gli utenti. I servizi potrebbero essere a pagamento o gratuiti. I cloud pubblici possono aiutare anche le aziende facendo risparmiare sui costi di gestione e manutenzione di un cloud.
- Cloud ibrido: è la combinazione tra pubblico e privato ,consente alle aziende di scalare la propria infrastruttura facendo pagare a queste solo le risorse che

utilizzano. In pratica alle aziende viene offerto il vantaggio di un cloud pubblico con la sicurezza di un cloud privato.

Un ulteriore divisione del cloud computing è in base al servizio offerto. L'infrastruttura è divisa tra tre tipi :

- Iaas: infrastructure as a service, in questa tipologia il fornitore è il responsabile della fornitura di server e archiviazione. L'utente finale non si occupa della gestione dell'infrastruttura avendo il controllo sull'archiviazione e sulle applicazioni distribuite;
- Paas: Platform as a service, in questa tipologia l'utente ha a disposizione un ambiente di sviluppo in cui si può sviluppare ed eseguire applicazioni senza interessarsi dell'infrastruttura. Agli utenti non viene chiesto di gestire l'infrastruttura intesa come server , rete, etc. Gli utenti devono concentrarsi esclusivamente sulle loro applicazioni, l'infrastruttura è gestita dai fornitori.
- Saas: software as a service, in questa tipologia all'utente è consentito accedere al software di un provider allocato sul cloud. I software non devono essere scaricati e nemmeno installati, ma sono accessibili tramite web. In questo modello è tutto gestito dal provider.

### **1.3.2 Vantaggi e limiti del cloud computing**

I vantaggi del paradigma cloud, che ne hanno permesso l'ampia e rapida diffusione sono molteplici , i più importanti sono :

- Riduzione dei costi: la manutenzione dei sistemi è complessa e costosa, però grazie a questo paradigma la situazione cambia permettendo anche a realtà molto piccole di avere sistemi IT interessanti per le loro attività, pagando solo per i servizi di cui necessitano;
- Scalabilità: il cloud consente alle organizzazioni di aumentare la platea di clienti in tempi brevi;

- Accessibilità: il poter accedere ai servizi cloud da qualsiasi zona agevola le organizzazioni e permette di gestire anche eventuali interruzioni non volute. Inoltre garantisce archiviazione e protezione dei dati;

D'altro canto il cloud computing non è perfetto e ci sono ancora delle sfide da superare, ad esempio anche se il provider garantisce sicurezza sull'integrità dei dati, attori malevoli preferiscono l'uso del cloud ad un'archiviazione locale perché aumentano le probabilità di intercettazione di questi dati. Un altro problema è un accesso limitato rispetto ad avere ad esempio un server proprio da gestire, mentre nel caso del cloud computing la gestione e l'organizzazione del sistema è a carico del provider.

Oltre questi problemi, esistono anche i problemi collegati all'evoluzione del mondo IoT e quindi anche di quello che ne deriva come le città intelligenti. Come è stato già detto nei paragrafi precedenti, una smart city genera una grande quantità di dati attraverso i dispositivi intelligenti che operano nelle città. Pensare di gestire i dati su un singolo data center centrale e distante fisicamente dal luogo non permetterebbe sviluppi in scenari in cui è richiesta una bassa latenza e un'analisi in tempo reale. Inoltre anche il cloud ha un limite nella potenza di calcolo e potrebbe saturare velocemente in certe situazioni in cui si potrebbe ritrovare. Un nuovo approccio è quello distribuito e decentralizzato che prende il nome di edge/fog computing, modello che prevede una gestione dei dati più vicina fisicamente ai bordi della rete, dove i dati vengono generati, in cui viene richiesta un'ottimizzazione dei dispositivi di bordo, per quanto riguarda l'edge computing; mentre per il fog computer si cerca di avvicinare la struttura dei cloud ai bordi della rete andando a replicare, limitatamente, la struttura dei cloud permettendo così una gestione più rapida dei dati, e in molti casi senza interagire con il cloud. Questo nuovo modello sarà analizzato approfonditamente nel prossimo capitolo.

# Capitolo 2

## Edge/Fog Computing

Nel seguente capitolo si espone il modello utilizzato per estendere il paradigma cloud. Con questo modello si introduce l'idea di avvicinare le risorse computazionali alla sorgente di dati, Edge e fog allargano l'orizzonte del cloud computing andando a distribuire la tecnologia fino ai bordi della struttura (da qui il nome edge) permettendo così una maggiore flessibilità all'utente finale, quindi non è un rimpiazzo al modello del cloud ma un'estensione di esso suddivisa su più layer.

### 2.1 Introduzione edge/fog computing

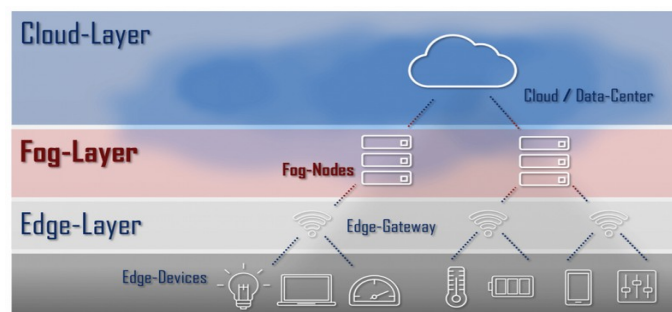
Per avvicinare il cloud computing all'utente finale è utilizzato questo paradigma di edge/fog computing che permette di distribuire le risorse e portare maggiore efficienza in diversi scenari, permette anche di limitare la latenza, un fattore importantissimo e non trascurabile per quanto riguarda le smart city, ma anche alcune applicazioni che in futuro ritroveremo in esse, come la guida autonoma che necessita di analisi dei dati e delle risposte corrette in un tempo massimo di un decimo di secondo. Se nel cloud la capacità di calcolo è concentrata tutta sul server centrale, ed è una potenza non sostituibile totalmente, che risulta essere lontano km dall'utente finale, attraverso questo paradigma le distanze si accorciano totalmente, per questo motivo entrano in campo edge e fog computing..

L'edge si realizza proprio dove i dati vengono generati, all'estremità della rete, anche per questo il nome edge(bordo). Fin qui però non si nota la loro peculiarità, dobbiamo sapere che come detto precedentemente i dati non possono essere inviati tutti al cloud a spese del tempo, soprattutto in scenari in cui la risposta è richiesta nel decimo di secondo, come nel campo della guida autonoma e delle smart city. I sensori, le telecamere che fanno parte dell'edge pre-elaborano i dati al momento dell'acquisizione, filtrandoli e in alcuni casi possono anche restituire una risposta concreta, ma in altri casi necessitano di una potenza di calcolo maggiore che un semplice nodo edge non può sostenere. Entra in gioco qui il livello intermedio tra edge e cloud, ovvero il fog, rappresentato molte volte da stazioni

sparse per le città, elaborano i dati e inviano al data center i dati necessari. I dispositivi di fog possono filtrare ulteriormente i dati da inviare al cloud oppure elaborarli loro direttamente e fornire una risposta ai dispositivi. Il layer di fog può quindi avere un grosso impatto sulla latenza e sulla mole di dati da spostare, anche in base alle esigenze di accesso della posizione. Il fog quindi filtra o analizza direttamente i dati generati e già pre-elaborati dall'edge, e decide quindi se può rispondere direttamente alle esigenze o se filtrare e mandare al cloud. Basti pensare ad un servizio di sicurezza composto da telecamere, che generano immagini continuamente, e che sono per la maggiore immagini ininfluenti e quindi dati scartabili. Pensare che questi dati vengano forniti continuamente ad un cloud è un problema, per questo con un filtraggio da parte dei nodi fog si possono risolvere i problemi direttamente dando anche risposte più rapide, ma anche da un punto di vista di sicurezza informatica abbiamo un limite a intercettazioni malevole, perchè più dati mettiamo in rete e più è alta la probabilità che questi dati vengano appunto intercettati. Ovviamente cloud è fondamentale perchè mette a disposizione una potenza di calcolo che non è riscontrabile su un comune computer. Cloud che per aziende fornisce servizi molto complessi e che riceve dati complessi, basti pensare al mondo dell'IoT. I dati devono quindi passare dal dispositivo fisico al cloud, che però in scenari come quello delle smart city non possono, ed è per questo che sono nati l'edge e il fog computing, che devono combattere il problema della latenza per dei servizi che necessitano l'invio di dati e il ritorno dei dati elaborati in pochissimo tempo.

### 2.1.1 Suddivisione su più livelli

Dopo aver visto l'implementazione in scenari reali dell'edge/fog computing, possiamo anche affermare che insieme al cloud computing creino una piramide, suddivisa in tre livelli.



*Figura 5: Rappresentazione layer cloud, fog/edge computing*

Fonte: <https://vitolavecchia.altervista.org/caratteristiche-e-differenza-tra-cloud-fog-e-edge-computing/>

La visione piramidale, o a livelli, permette anche di capire al meglio la scalabilità di questo paradigma, infatti la gestione efficiente di miliardi di dati non sarebbe possibile se a gestirli fosse un livello singolo, essendo un paradigma distribuito permette la gestione e l'analisi dei dati fin dai bordi, ovvero dai dispositivi che generano i dati.

- Cloud computing: i dati vengono elaborati su un server cloud centrale e distante dalle fonti di informazioni;
- Fog computing: livello “nebbia”, si interpone tra cloud ed edge e ,interfacendosi direttamente con i dispositivi di edge, collegati nella rete LAN, i cui dati sono elaborati e restituiti se i tempi sono limitati, oppure se richiedono elaborazioni complesse vengono inviati al datacenter centrale per l'elaborazione;
- Edge computing: è il livello più basso, ne fanno parte tutti i dispositivi dotati di tecnologia IoT, è dove i dati vengono generati e molte volte già elaborati direttamente dal dispositivo per operazioni basilari data la limitata potenza di calcolo, oppure inviati al fog dopo averli filtrati;

Quindi il cloud computing è più adatto ad analisi più precise e approfondite , edge/fog sono più adatti ad analisi veloci per risposte immediate. Edge/fog possono anche funzionare senza internet a differenza del cloud che richiede connessioni continue; inoltre nel fog i dati restano tra i nodi distribuiti mentre sull'edge restano direttamente sui dispositivi quindi è un'ulteriore sicurezza su eventuali azioni malevole.

Infine è bene definire che la divisione piramidale è prettamente teorica, infatti nella pratica i livelli sono 4 come vediamo bene dalla *Figura 3*, nello specifico si differenzia l'edge computing con i dispositivi edge (sensori) . Questa precisazione ,però, non cambia l'obiettivo finale di garantire una distribuzione adeguata delle risorse permettendo così di limitare la latenza garantendo una capacità di calcolo adeguata alle diverse esigenze.

## **2.2 Fog computing**

Dopo aver dato un'ampia panoramica al paradigma Edge/Fog definiamoli nello specifico. Il Fog computing è stato coniato da Cisco . É una nuova tecnologia che offre vantaggi al mondo dell' "informatica delle cose" , non nasce come sostituto del cloud computing ma come un'espansione di esso . Infatti fornisce servizi simili dall'archiviazione all'elaborazione dei dati , evitando così di dover comunicare costantemente con il cloud ma avendo quindi una sua capacità di calcolo ed analisi rispondendo in tempi brevi ai nodi edge riducendo così la latenza. L'utilizzo del fog computing in ambiti IoT , quindi anche Smart city crea nuovi servizi come i FaaS, in cui un fornitore, nel nostro caso chi gestisce lo sviluppo delle Smart City , decide di distribuire diversi nodi Fog, i quali avranno una propria potenza di calcolo e archiviazione. Questo nuovo servizio permette anche a piccole realtà , inteso come aziende medio-piccole di poter avere servizi di calcolo e archiviazione cosa che con i data center cloud non era per niente scontato .

C'è anche da dire che è un paradigma con capacità limitate rispetto al cloud classico , sia per la capacità di calcolo che per la capacità di archiviazione.

### **2.2.1 Caratteristiche del Fog Computing**

Il fog computing come visto nel precedente paragrafo è un 'estensione del cloud computing, però con una vicinanza maggiore ai nodi di bordo , alle fonti dei dati fornendo però gli stessi servizi del cloud con delle limitazioni sulla capacità di calcolo e archiviazione. Come per il cloud i nodi di Fog sono possono essere tutti questi dispositivi con una connessione che allo stesso tempo offrono elaborazione ed archiviazione dei dati quindi server, router, switch e/o controller industriali.

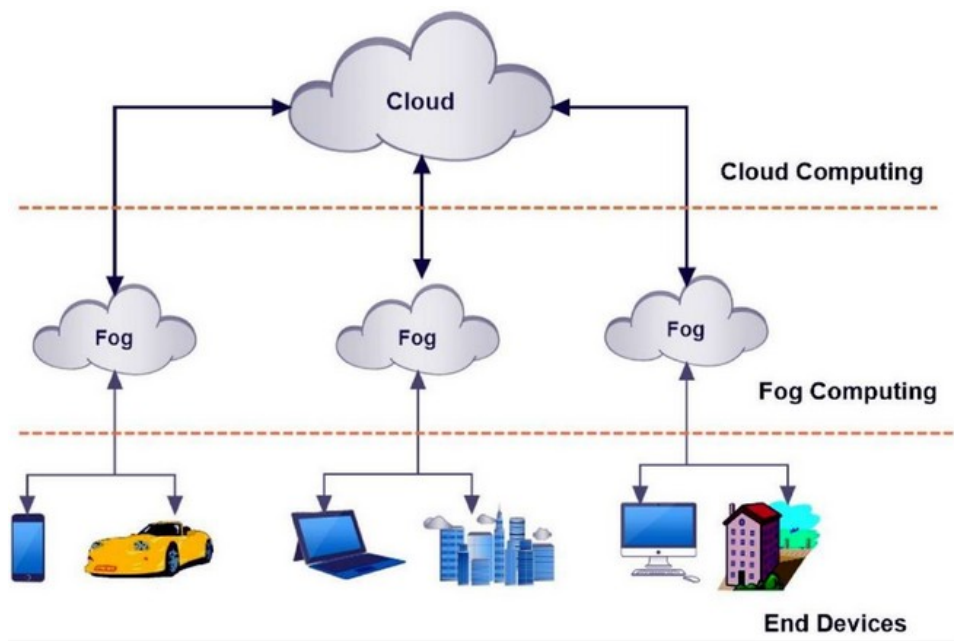


Figura 6: Fog visto come estensione del cloud ma più vicino ai dispositivi finali  
 Fonte: <https://www.mdpi.com/2504-2289/2/2/10>

Le caratteristiche principali del fog computing possono essere racchiuse in :

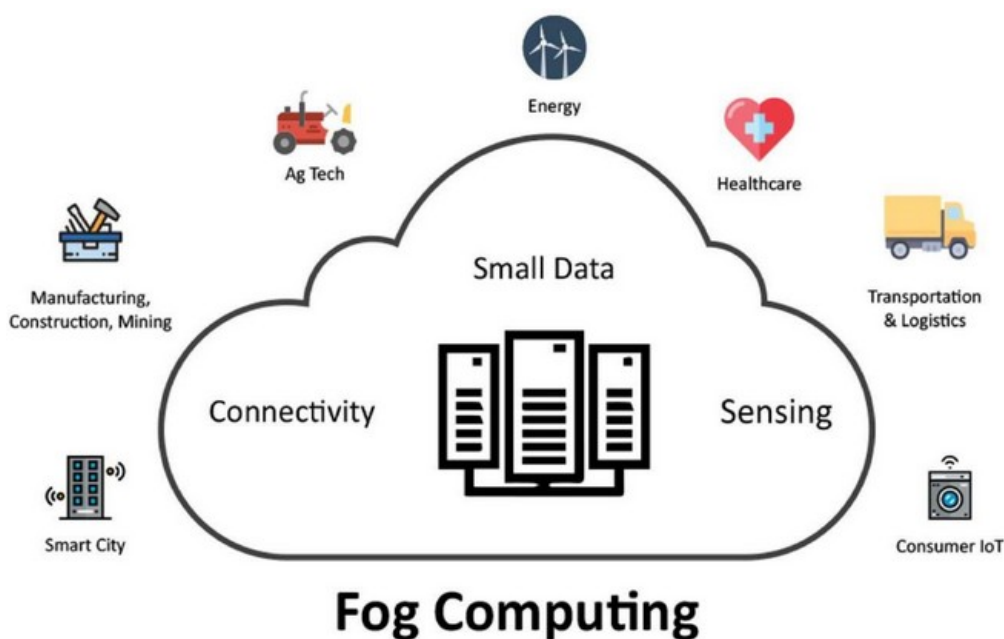
- **Bassa latenza :** la vicinanza alle fonti dei dati e la possibilità di avere diversi nodi fog distribuiti fornisce una latenza inferiore;
- **Distribuzione:** come detto in precedenza, il fog computing differisce dal cloud computing, nonostante l'utilizzo di tecnologie praticamente uguali, proprio per la possibilità di distribuire diversi nodi fog ovunque;
- **Scalabilità:** caratteristica che deriva dalla possibilità di distribuire più nodi fog così da avere reti di nodi che possono gestire i dati senza creare troppe code;
- **Supporto alla mobilità:** permettere direttamente ai dispositivi mobili di comunicare con i nodi nebbia .

Quindi da queste caratteristiche possiamo anche definire il funzionamento . Un numero elevato di dispositivi distribuiti geograficamente, dispositivi che abbiamo visto possono essere di qualsiasi tipo ,inclusi router,switch in modo da formare un cloud in miniatura ai margini della rete. Questi dispositivi si gestiscono in autonomo andando ad applicare quella che è l'idea di fondo del fog computing , ovvero di caricare o scaricare i dati da e verso la rete principale. I dispositivi di bordo che generano dati , dopo averli pre elaborati li mandano al fog che può già rilasciare risorse come archiviazione ed elaborazione senza



scomodare il cloud principale. Solo se questa attività è al di fuori delle capacità dei nodi fog, allora i dati vengono inviati al cloud principale per permettere l'elaborazione. Così facendo si riduce il carico verso il cloud da cui derivano tutti i benefici visti in precedenza. Nello specifico, ad esempio, se i dati provenienti dai nodi di bordo sono tutti correlati tra loro, allora il nodo di fog non dovrà caricarli tutti nel data center ma caricherà solo i dati di una singola entità edge, riducendo così i tempi. Allo stesso modo, se i dati provenienti dal cloud sono correlati per un gruppo di entità edge, allora lo inoltrerà solo ad una singola entità permettendo poi la condivisione nello stesso gruppo edge delle informazioni. Inoltre, a questo viene anche combinato l'utilizzo di cache che in cui vengono caricate le informazioni che riguardano attività più popolari, permettendo così di ridurre ulteriormente il tempo.

Grazie alla diffusione di questo paradigma riusciamo oggi a parlare di città intelligenti, e nello specifico di progetti come la guida autonoma, che senza elaborazioni di dati in tempo reale non potrebbe realizzarsi, cosa non applicabile a scenari in cui tutti i dispositivi comunicano con un singolo data center.



*Figura 7: Fog computing e applicazioni IoT*

Fonte: <https://www.mdpi.com/2504-2289/2/2/10>

Il fog computing si rivela un ottimo paradigma, nonostante sia abbastanza, infatti è argomento di ricerche che discutono la convergenza del mondo IoT con il fog computing.

### **2.2.2 Applicazioni del Fog computing con Smart City**

Esistono molte aree in cui il fog computing svolge un ruolo fondamentale, tra queste ovviamente c'è la mobilità urbana intelligente. Nel capitolo precedente sono stati definiti alcuni servizi che potrebbero trarre vantaggi dal uno sviluppo intelligente della mobilità, tra cui la stessa mobilità pedonale. Sempre nel capitolo precedente sono stati definiti i veicoli collegati con le infrastrutture (V2I) oppure veicoli connessi tra di loro (V2V):

- Auto connesse: l'obiettivo finale è quello della guida autonoma, a cui si arriverà ovviamente dopo diversi step. Uno dei primi sarà sicuramente quello di permettere alle auto di connettersi tra di loro (V2V) e ovviamente con internet. Per realizzarsi ovviamente il fog computing avrà un ruolo centrale poiché c'è la necessità di interazioni, quindi automaticamente gestione dei dati, in tempo reale. Questo porterà sicuramente un aumento della sicurezza stradale, prevedendo ed evitando incidenti con frenate assistite, grazie anche alla comunicazione con le infrastrutture (V2I);
- Semafori intelligenti: tra le infrastrutture che dovranno interagire con i veicoli ci saranno sicuramente i semafori, dotati di telecamere e sensori, comunicanti con i veicoli attraverso sempre i nodi fog. Rileveranno la presenza di pedoni e ciclisti andando ad evitare collisioni con i veicoli.

Questi sono solo alcuni dei miglioramenti alla salute e al traffico che la combinazione tecnologia ICT e software porteranno nei centri urbani, rendendoli quindi a tutti gli effetti smart.

### **2.2.3 Vantaggi e svantaggi del Fog computing**

Tra i principali vantaggi del Fog computing rientrano la maggior parte delle sue caratteristiche.

Riduzione della latenza, che permette un'accelerazione nell'elaborazione e nell'analisi del dato. La possibilità di distribuire i nodi e dispositivi che poi saranno i nodi fog, da qui la scalabilità e meno code che potrebbero saturare la rete. La possibilità di combinarlo con l'uso di cache che velocizzano ulteriormente la gestione dei dati, o un archiviazione limitata dei dati.

Gli svantaggi però ricadono sui costi dell'infrastruttura da creare e anche sui possibili rischi sulla sicurezza informatica dovuti anche al non poter applicare gli stessi metodi utilizzati per i cloud, essendo i nodi di fog diversi in caratteristiche, soprattutto per la mobilità e la diffusione su larga scala. Infatti i nodi di fog risultano essere degli obiettivi interessanti per attori malevoli perché grazie alla possibilità di archiviazione dei dati, essi contengono dati sensibili che provengono anche dal cloud principale, per questo sono necessarie altre ricerche per migliorare questo aspetto di sicurezza. Una possibile soluzione ai problemi di sicurezza è quello dell'implementazione del paradigma di edge computing che limiterebbe ancora di più la circolazione dei dati tra i 3 livelli, attuando le prime fasi di elaborazione e/o filtraggio dei dati direttamente sul dispositivo generatore.

## **2.3 Edge Computing**

Dal nome possiamo intuire la posizione in cui opera questo paradigma. Edge, o meglio bordo in italiano, definisce il paradigma che opera ai margini della rete. È il punto più vicino ai dispositivi che generano i dati, invece che in luoghi distanti fisicamente come lo sono i nodi fog ad esempio.

Con l'utilizzo del paradigma edge la latenza si riduce ulteriormente perché i dati non saranno inviati quindi automaticamente si elimina la necessità di attenderne il ritorno, aumentando anche l'efficienza del servizio. In questo modo si riducono anche i costi e allo stesso tempo permette ai dispositivi di operare anche con una connessione non fissa, inoltre evitando un inoltro dei dati, attraverso l'elaborazione di quest'ultimi sul dispositivo stesso, aumenta la sicurezza evitando attacchi malevoli. Le somiglianze tra Edge e Fog sono molteplici, perché effettivamente svolgono compiti simili, ma è anche vero che l'edge computing può esistere senza il fog ma non viceversa, questo varia in base agli ambiti e ai calcoli richiesti. Inoltre l'utilizzo del paradigma edge/fog computing permette di velocizzare ulteriormente le operazioni; basti pensare ad una zona di "nebbia" che deve

elaborare dei dati che ha già preelaborato perché simili ad altri ricevuti, e questo può farlo senza inoltrarli al cloud come farebbe un nodo edge in caso ovviamente di dati non processabili direttamente sul dispositivo di bordo. Partiamo con un esempio, dato da una telecamera di sicurezza che è realizzata per registrare continuamente immagini che però nella maggior parte del tempo sono scartabili. Se questo dispositivo di edge dovesse inviare sempre queste immagini al cloud centrale porterebbe il cloud a saturare, ma inviandoli al nodo di fog più vicino permetterebbe un filtraggio prima di un inoltro al data center principale.

### **2.3.1 Differenze e somiglianze con il Fog Computing**

Dopo aver parlato del paradigma edge/fog computing si nota subito una certa somiglianza tra edge e fog. Entrambi offrono una sicurezza ai dati, escludendo ovviamente il caso di “men in the middle” che si potrebbe verificare quando avviene una condivisione dei dati. Si è visto come risolvono anche problemi di connessione non stabile, permettendo ai dispositivi di continuare ad operare ed elaborare i dati localmente. Queste somiglianze però non devono confondere e far pensare che siano dipendenti l’uno dall’altro, infatti se è vero che il fog computing non potrebbe esistere senza dispositivi edge, perché i nodi fog non generano dati ma elaborano quelli che ricevono dai dispositivi stessi; è anche vero che l’edge computing può esistere senza fog. Questa particolarità avviene semplicemente perché i nodi edge hanno capacità di gestione ed elaborazione dei dati e possono analizzarli sul sensore o dispositivo stesso. I dati, quando si utilizza il paradigma edge, non vengono trasferiti riducendo i costi e permettendo così analisi in tempo reale, garantendo anche la sicurezza di integrità. Inoltre l’edge qualora lo volesse potrebbe inviare direttamente i dati al cloud. Spontaneo sarebbe chiedersi il perché dell’uso del fog computing, ma la risposta sarebbe scontata; la potenza di calcolo di un dispositivo edge è limitata e non potrebbe raggiungere quella di un nodo fog che abbiamo visto essere una replica in parte del cloud, quindi con una capacità maggiore. Infatti l’uso singolo del paradigma edge è limitato a servizi come il monitoraggio sanitario di pazienti in tempo reale o giochi multiplayer. D’altra parte il connubio edge/fog è utilizzato per scenari in cui si devono gestire grandi quantità di dati come Smart city e quello che ne deriva, come rappresentato nella successiva figura.

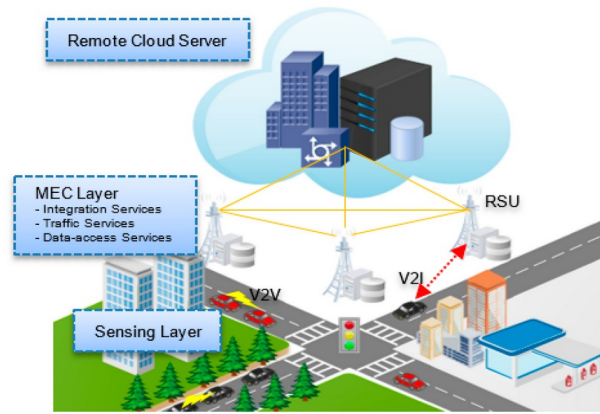


Figura 8: Rappresentazione della divisione a livelli edge/fog in una città intelligente.

Fonte: <https://www.mdpi.com/1424-8220/19/5/1073#>

### 2.3.2 Esempi di scenari applicativi di edge computing e dispositivi diffusi

Attualmente il paradigma edge computing rappresenta un aspetto chiave per la nascita di smart city.

Due esempi sono :

- Segnalazione di incidenti : in un futuro in cui le auto saranno autonome, definiti come di livello 5. La guida autonoma è uno scenario complesso, basti pensare che un semplice cambio corsia potrebbe portare ad una collisione. Queste analisi devono essere svolte in tempo reale che potrebbero essere svolte da unità a bordo strada abilitate all'edge computing, per una tempestiva segnalazione dell'incidente per un tempestivo intervento dei soccorsi . Infatti in un eventuale incidente, in cui potrebbero esserci dei feriti, necessita dell'intervento di medici, sanitari e forze dell'ordine che in caso di incidente grave dovrebbero essere avvertite da queste unità di bordo che attraverso algoritmi devono valutare la gravità dell'incidente, tutto a bassa latenza.
- Parcheggio intelligente: I sistemi di parcheggio più diffusi soffrono problemi legati alla gestione dei parcheggi e ad un elevato tempo di calcolo per trovare parcheggi vuoti. Una semplice gestione di questi attraverso l'edge computing, potrebbe risolvere questi problemi con un calcolo istantaneo, nello specifico tramite l'utilizzo

di telecamere intelligenti che esaminano le immagini attraverso software di intelligenza artificiale per trovare i parcheggi vuoti.

Questi sono solo due esempi di implementazione nelle città di dispositivi intelligenti utili per raggiungere gli obiettivi delle città intelligenti. Tra le altre applicazioni dell'edge computing rientrano anche :

- Gestione del traffico nelle città.
- Monitoraggio remoto delle risorse, come installazioni di petrolio e gas
- Case intelligenti che soddisfano le esigenze dei loro occupanti.

Per realizzare questi scenari, come già è stato detto in precedenza c'è bisogno di dispositivi intelligenti che raccolgono ed elaborano, anche parzialmente i dati . Tra i più diffusi :

- Dispositivi Internet of Things;
- Altoparlanti intelligenti;
- Telecamere di sicurezza;
- Smartphone;

Questi sono solo alcuni dei tanti dispositivi che ad oggi permettono ad alcune realtà di diventare intelligenti grazie al paradigma edge.

### **2.3.3 Evoluzione dell' Edge computing**

L'unione tra Fog ed edge computing, quando quest'ultimo è composto da dispositivi mobili , dà vita al mobile edge computing. L'idea è quella di portare le offerte del cloud sui dispositivi finali , il MEC può essere visto come un server cloud installato sui dispositivi mobili, permettendo così una velocizzazione delle risorse di calcolo e archiviazione. IN realtà però con utenti finali si intende i provider , infatti il MEC è pensato per i provider di servizi. Il disaccoppiamento tra servizi e dispositivi fisici permette ai provider di gestire al meglio il carico di lavoro derivante da più ecosistemi. Molti provider scelgono di spostare i servizi sempre più verso il bordo della rete, precisamente nelle sedi di rappresentanza, rendendo così il servizio più rapido abbassando ulteriormente la latenza. Ad esempio, quando un dispositivo connesso a una rete 4G si collega alla rete mobile di un provider di servizi di telecomunicazioni, la maggior parte delle applicazioni mobili, si trova in una posizione centrale, in data center distanti fisicamente dall'utente. Con l'architettura MEC i provider possono permettersi di avvicinare i carichi di lavoro dei dispositivi mobili all'utente.

### **2.3.4 Vantaggi e svantaggi dell'edge computing**

I benefici dell'edge computing risultano essere molto simili a quelli del fog computing, tra i diversi benefici rientrano :

- Riduzione della latenza;

- Efficienza operativa migliorata;
- Costi di larghezza di banda ridotti.

Allo stesso modo però il paradigma edge non è perfetto e comporta rischi e sfide da superare.

Avendo un ampliamento della rete su più nodi, si amplia anche la platea di attori malevoli interessati ai dati, maggiori sfide quindi per la sicurezza. Inoltre una problematica, già accennata precedentemente, c'è la capacità di calcolo limitata rispetto al cloud che resta quindi non rimpiazzabile totalmente.



# Capitolo 3

## SUMO

Nel precedente capitolo sono stati esaminati gli aspetti teorici dell'edge/fog computing e alcuni scenari applicativi reali. In questo capitolo saranno illustrati aspetti tecnici per effettuare simulazioni di scenari utili per l'urban smart mobility, scenario centrale nello sviluppo di una Smart City . Dunque sarà discusso un ambiente di simulazione, sarà analizzato uno scenario e discussi alcuni tool utilizzati per l'analisi e la generazione di scenari realistici.

### 3.1 Descrizione del Problema

Come analizzato e descritto nei precedenti capitoli, la costante crescita e diffusione di sistemi di elaborazione e dispositivi intelligenti è il principale punto di riferimento per lo sviluppo del paradigma edge/fog computing. Sempre nei precedenti capitoli sono stati descritti alcuni degli scenari nei quali il paradigma edge/fog computing è fondamentale, molti dei quali sono parte fondamentale per la realizzazione di una smart cities. Lo scenario tipico di una smart city comprende ovviamente sensori che generano una grande quantità di dati che vengono inviati ai nodi fog ( come è stato detto in precedenza lo scenario di una smart city non rientra negli scenari che possono implementare solo uno dei due paradigmi edge o fog ma richiede che entrambi siano utilizzati, per mettere ad alcuni sotto-scenari delle smart city come ad esempio la guida autonoma di avere un'analisi dei dati in tempo reale) che devono elaborarli e inviarli al cloud centrale, dove possono essere elaborati ulteriormente e archiviati. Altri scenari nelle smart city vengono comunque elogiati sviluppi che mirano al miglioramento del traffico oppure ad un controllo intelligente dei parcheggi, oppure anche al controllo della mobilità pedonale, analisi importante per realizzare anche scenari di guida autonoma nelle città intelligenti. Quindi restando in quest'ottica è chiaro che per uno sviluppo sostenibile di analisi del traffico urbano è necessario integrare qualsiasi tipo attore nella progettazione, una progettazione che molte volte necessita di simulazioni realistiche prima di essere applicate in scenari reali.

### **3.1.1 Obiettivi, tecnologie utilizzate e ambiente di lavoro**

Dopo aver descritto il problema da affrontare, o meglio da simulare, si definiranno gli obiettivi, le tecnologie utilizzate e l'ambiente di lavoro utilizzato per completare questo tipo di studio.

La richiesta in ottica della mobilità veicolare e pedonale è quella di studiare un simulatore ,Sumo, in grado di effettuare la riproduzione di tracce reali , con la possibilità di definire il comportamento dei pedoni/veicoli. L'obiettivo è quello di poter studiare scenari reali attraverso il simulatore Sumo, che sarà argomento della sezione 3.2.

Quindi, lo studio è stato effettuato completamente su Sumo, Simulation of Urban Mobility, mentre per la creazione e l'analisi delle tracce realistiche si è utilizzato alcuni tools scritti in Python, in grado di creare una traccia compatibile con il simulatore, utilizzando Open Street Map, e altri tools che permettono di popolare realisticamente queste tracce.

## **3.2 Simulation of Urban MObility**

Sumo, Simulation of urban mobility, è un pacchetto di simulazione di mobilità urbano open source e multimodale, sviluppato dal centro aerospaziale tedesco. La caratteristica di essere un pacchetto open source ha permesso un costante sviluppo che ha portato Sumo a non essere solamente un semplice simulatore, ma consente attraverso diversi applicativi di gestire anche queste simulazioni. Un'altra caratteristica importante è la simulazione “multi-modale”, in particolare definisce un tipo di simulazione non limitata esclusivamente alle autovetture ma anche altre tipologie di attori, che possono essere veicoli per il trasporto pubblico o semplicemente dei pedoni e via dicendo. La novità di questo pacchetto di simulazione è il poter simulare nuove strategie di traffico prima che queste vengano utilizzate in situazioni reali. Le prime versioni sviluppate erano molto leggere e permettevano l'utilizzo solo da terminale. Ad oggi è fornita una interfaccia grafica che permette di ricreare delle reti a piacere e di poter modellare comportamento di ogni entità a piacere attraverso le impostazioni presenti. Essendo diviso in più parti, il software è unico e

lo diversifica da altri simulatori, come ad esempio l'assegnazione delle entità che qui è compito di un'applicazione esterna, netedit.

### 3.2.1 Creazione di reti

La creazione delle reti in Sumo è possibile in diverse modalità. La più semplice è la creazione attraverso l'interfaccia grafica di Sumo stesso. Infatti è possibile tramite il pacchetto Sumo poter generare delle reti non reali, che rappresentano degli scenari non ancora realizzati nel complesso, ed è possibile tramite netedit. Netedit è una delle applicazioni esterne, nello specifico è un editor di rete visiva ed è utilizzato principalmente per creare reti da zero e il perché sarà chiaro tra poco. Netedit si basa su netconvert, che è un'altra applicazione avviabile da linea di comando che permette di trasformare di importare reti stradali da diverse fonti. Per i file da convertire con netconvert sono necessari file xml, uno per i nodi e uno per le strade. Tornando all'applicazione netedit, è un'applicazione più comoda da utilizzare, soprattutto per i novizi perché permette di svolgere tutti i task di netconvert ma attraverso un'interfaccia grafica molto chiara, che è possibile visionare nella figura seguente:

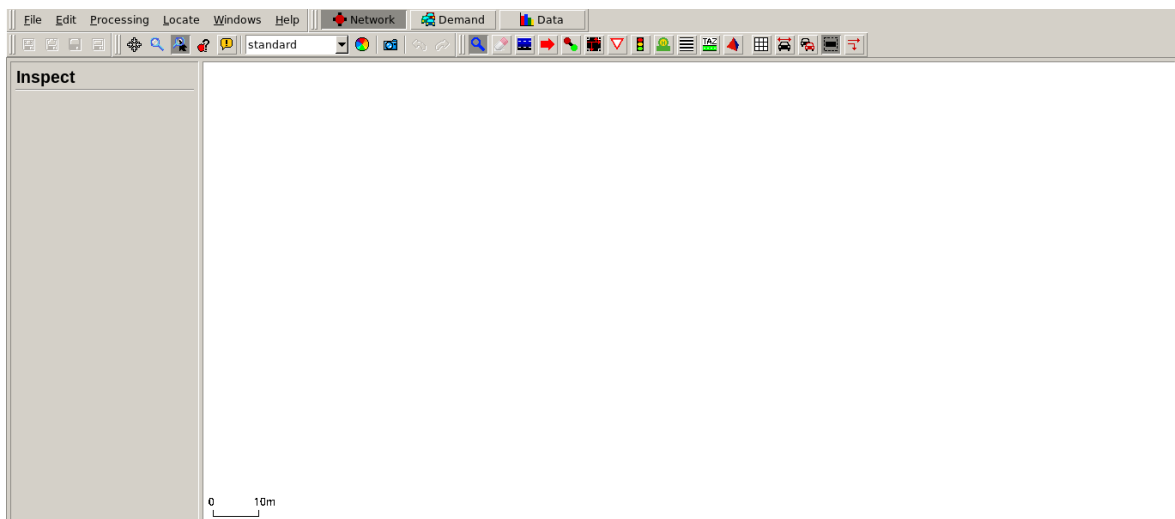


Figura 9: Netedit interfaccia grafica

Da questa interfaccia è possibile, molto intuitivamente, selezionare la modalità Edge per poter creare dei nodi. Una volta creati i nostri nodi, andando in modalità Demand sarà possibile definire dei percorsi (Route), cliccato Route → e successivamente sui bordi creati.

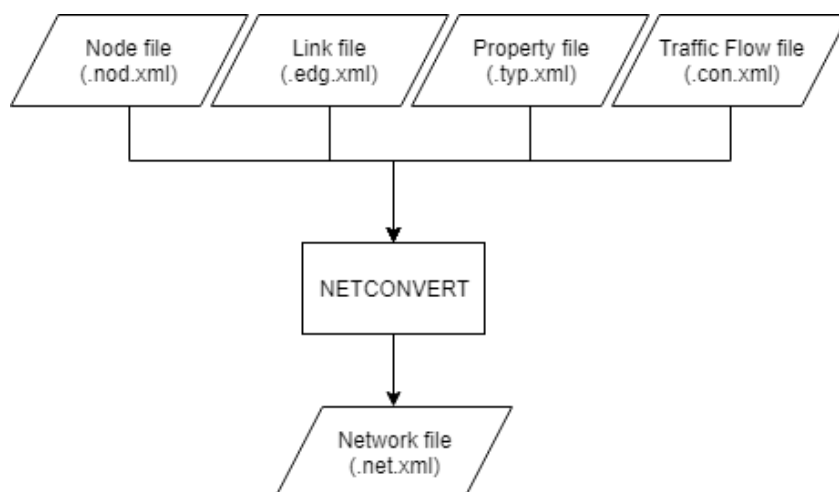
Definita la route, sarà possibile attuare ulteriori modifiche sul tipo di percorso e sui suoi limiti. Ad esempio un percorso può essere accessibile da tutti o limitato esclusivamente ai pedoni, o ai soli veicoli di un determinato tipo (ad esempio autovetture o bus del trasporto cittadino). Infine, dopo aver definito il percorso e la rete, per poter visualizzare la simulazione creata è necessario salvare la configurazione, per utilizzi futuri, ed è necessario aprirla su sumo-gui, l'interfaccia grafica principale. I file generati saranno:

- file.net.xml: definisce la rete creata;
- file.rou.xml: contiene il percorso creato e l'entità registrata su quel percorso (tipologia di veicolo o pedone);
- file.sumocfg: file di configurazione generale, contiene la simulazione completa. Se letto, questo file contiene un richiamo sia a file.net.xml che a file.rou.xml.

Una volta salvata la rete, successivamente sarà necessario eseguire file.sumocfg, dopo averlo aperto tramite sumo-gui, per ripetere la simulazione specifica.

Ovviamente questi file possono essere modificati anche tramite un editor, qualora fosse necessario.

Inoltre, come detto in precedenza, è possibile definire più file, che tramite netconvert costituiranno la rete:



*Figura 10: Input per creazione rete tramite Netconvert*

*Fonte: [https://www.researchgate.net/figure/Conversion-of-network-file-using-Netconvert\\_fig4\\_368943366/download](https://www.researchgate.net/figure/Conversion-of-network-file-using-Netconvert_fig4_368943366/download)*

Come, comprensibile dalla Figura 10, per definire una rete tramite netconvert è necessario creare più file che definiranno :

- edge: file.edg.xml, contenente le informazioni sui bordi (roads/streets);
- type : file.typ.xml, contenente le definizioni dei tipi di bordi;
- node: file.nod.xml, contenente le informazioni dei nodi;
- connection: file.con.xml, descrive le connessioni tra bordi in ingresso e uscita;

Dopo aver definito questi file , si può procedere con netconvert da linea di comando per assemblarli e ricavare il file.net.xml :

```
netconvert --node-files=test.nod.xml --edge-files=test.edg.xml --type-files=test.typ.xml --connection-files=test.con.xml --output-file=test.net.xml
```

*Figura 11: Esecuzione di netconvert da linea di comando*

Dopo aver ricavato il file.net.xml è ora possibile seguire più strade per giungere ad una simulazione. Ovviamente il file di rete da solo non basta, serve un popolamento di questa rete che solitamente, tramite interfaccia grafica di netedit è semplice da fare. Una volta compresi meccanismi basilari per la creazione di una rete semplice bisogna anche definire la creazione di reti complesse. Infatti in questi file, nella maggior parte dei casi di creazione reti reali necessitano dell'aggiunta di alcuni dettagli , come possono essere semafori o attraversamenti pedonali che devono essere definiti in questi file (file.nod.xml). Oppure la definizione del senso di marcia di alcune strade (file.edg.xml), o ancora limiti di velocità per un determinata classe di veicoli. Quindi è ben comprensibile che più è dettagliata la traccia che si vuole ricreare o creare , e più sarà arduo ricavare la rete tramite netconvert. Netconvert è utilizzabile anche per ricavare un file.net.xml da un OSM-file nativo , ricavato da Open Street Maps per simulazioni che vengono effettuate su scenari reali. In questo caso non è necessario creare i file tipo, nodo,bordo ,connessione e tutte le specifiche che li compongono , perché tramite Open Street Maps il file.osm.xml sarà completo delle specifiche di quello scenario.

Open Street Maps però è anche utilizzato da alcuni tools python che saranno analizzati successivamente, e che permettono all'utente di svolgere simulazioni su tracce reali senza interagire troppo con la riga di comando.

Un'altra applicazione, tramite la quale è possibile generare delle reti di diverso tipo, è NETGEN. Precisamente dopo aver appreso i parametri definiti dall'utente, permette di creare tre tipi di rete:

- griglia;
- ragnatela;
- casuale;

Nel caso a griglia, viene richiesto di inserire numero di incroci, la distanza tra essi e un file di output. Nel secondo caso viene creato un numero di perimetri stradali, divisi da un numero di archi, specificato. Infine, come deducibile dal nome, reti casuali.

Infine è giusto definire che le reti sono proiettate su un piano cartesiano 2D, e viene assegnata un'area che ha un punto inferiore, che nei casi di rete importate da tool è sempre corrispondente a  $(x,y) = (0,0)$  e un punto superiore che rappresenta la fine della rete costruita. Questi dati sono contenuti nel file di rete, all'elemento *<location/>* all'attributo *convBoundary*.

### 3.2.2 Entità della rete

Una volta definito il file.net.xml, per arrivare ad una simulazione reale è necessario popolare la rete con le giuste entità. Il file nello specifico è il file.rou.xml, un file nel quale si definiscono i tipi di entità e quali percorsi devono fare queste entità.

I percorsi variano in base al tipo di entità, vediamo qualche esempio:

```
<vehicle id="615_0" depart="triggered">
  <route edges="gneE0 gneE1"/>
</vehicle>
```

Figura 12: Definizione di un'entità nel file.rou.xml

Nella figura 12 viene definita un'entità di tipo veicolo con un determinato *id*, un punto di partenza che coincide con l'inizio del primo edge. Il bordo o i bordi, se l'itinerario dovesse prevedere più strade da percorrere, definiscono la route per il veicolo in questione. Inoltre i veicoli possono essere modificati con altri attributi, ad esempio:

- color;
- depart: la partenza può essere impostata in diverse modalità, quella vista è la modalità classica, altre potrebbero essere departPos da una certa posizione, departSpeed con la quale definiamo la velocità d'ingresso.
- Arrivo: punto di arrivo;
- personNumber: è possibile definire anche il numero di passeggeri del veicoli;

Altre interessanti modifiche per l'entità veicolo è il routing dinamico, infatti, Sumo permette di impostare questa modalità tramite la quale i veicoli calcolano il loro percorso dinamicamente tenendo conto dello stato del traffico della rete, quindi si adatta ad ingorghi e ad altri cambiamenti.

Un altro tipo di entità è l'entità persona:

```
<person id="613" depart="852.40">
  <walk edges="gneE0"/>
</person>
```

*Figura 13: Definizione dell'entità persona in file.rou.xml*

In quest'altra definizione, come è facile da intuire, viene definita una entità di tipo persona , questa entità ha anch'essa un id, un punto di partenza che nel caso di pedoni equivale a un punto preciso del bordo di partenza , che in questo caso è singolo, altrimenti equivarrebbe al punto del primo bordo della lista definita negli edges subito dopo. Si può notare nel caso dei pedoni che viene definita anche la modalità che assume il pedone sull'edge.

Nello specifico 3 le tipologie di comportamento di un pedone:

- walk: definisce il tipo pedone, se impostato su una persona nella simulazione, quest'ultima camminerà per tutto il suo percorso;
- stop: blocco temporaneo di un pedone sull'edge selezionato;
- ride: un pedone con questa modalità , cammina fino ad un punto in cui è disponibile un veicolo, definito precedentemente nel file, per poi salirci. In una simulazione reale rappresenterebbe uno scenario in cui sono presenti cittadini che utilizzano il trasporto urbano. Basterebbe definire come veicolo il tipo bus.

Un esempio dettagliato per l'entità persona con più comportamenti durante una simulazione lo vediamo nella figura seguente:

```
<routes>
  <person id="person0" depart="0">
    <walk from="2/3to1/3" to="1/3to0/3" departPos="80" arrivalPos="55"/>
    <ride from="1/3to0/3" to="0/4to1/4" lines="train0"/>
    <walk from="0/4to1/4" to="1/4to2/4" arrivalPos="30"/>
    <stop lane="1/4to2/4_0" duration="20" startPos="40" actType="singing"/>
    <ride from="1/4to2/4" to="3/4to4/4" lines="car0"/>
  </person>

  <vehicle id="train0" depart="50">
    <route edges="1/4to1/3 1/3to0/3 0/3to0/4 0/4to1/4 1/4to1/3"/>
    <stop busStop="busStop0" until="120" duration="10"/>
    <stop busStop="busStop1" until="180" duration="10"/>
  </vehicle>

  <vehicle id="car0" depart="triggered">
    <route edges="1/4to2/4 2/4to3/4 3/4to4/4" departPos="30"/>
    <stop lane="1/4to2/4_0" duration="20" startPos="40" endPos="60"/>
  </vehicle>
</routes>
```

Figura 14: Esempio di definizione multipla dell'entità persona in file.rou.xml

Nella figura14, che rappresenta il contenuto e la struttura classica di un file.rou.xml, è raffigurato un esempio di definizione entità persona con comportamenti multipli che , in questo caso , *person0* avrà. Inoltre come è possibile vedere sempre nella figura 14 , oltre la definizione di un'entità persona, se questa prevede l'utilizzo di un veicolo durante la simulazione allora vanno definiti anche i diversi tipi di veicoli che andrà ad utilizzare, specificandoli poi con **<ride from= .... to=.... lines=..... />** , che come è stato detto precedentemente , identifica il comportamento di un pedone che passeggia fino ad un punto di stop dove poi prenderà il veicolo associato.

Definite le entità che popolano le reti è corretto definire anche le tipologie di percorsi :

- trip: identifica il movimento di un movimento da una posizione ad un'altra , definito come un arco di partenza e uno di arrivo, arco inteso come strada;
- route: rimanendo sempre sulla figura14 , nella definizione del *vehicle* si può notare la voce **route**, serve per definire un un set di archi che il veicolo dovrà attraversare.



Per una simulazione realistica, è necessario che gli archi (strade) siano collegate tra di loro , altrimenti il simulatore non interromperebbe la simulazione ma sposterebbe, simulando un teletrasporto, il veicolo da una strada alla successiva.

D'altro canto è chiaro che scrivere un file.rou.xml da zero, per andare ad esempio a popolare una rete che rappresenta una grossa città , con una grande quantità di entità, è lento e noioso; ma fortunatamente il pacchetto Sumo è pieno di risorse.

Infatti come per la definizione della rete, per la quale c'è la possibilità di utilizzare Netedit, anche per il popolamento della rete è possibile utilizzare questa applicazione. Purtroppo, però, per una simulazione realistica il riempimento tramite interfaccia grafica è molto lenta. Come già accennato nel pacchetto Sumo è disponibile, come già visto per la creazione della rete con Netconvert, un applicativo DUAROUTER che calcola i percorsi più brevi tra quelli disponibili.

I percorsi disponibili sono inseriti in un file .xml , in cui definiamo i trips, che ricordiamo essere dei singoli archi , in questo caso strade. Definiti i file da utilizzare possiamo eseguire duarouter da linea di comando, che restituirà i percorsi ottimi calcolati tramite l'algoritmo di Dijkstra :

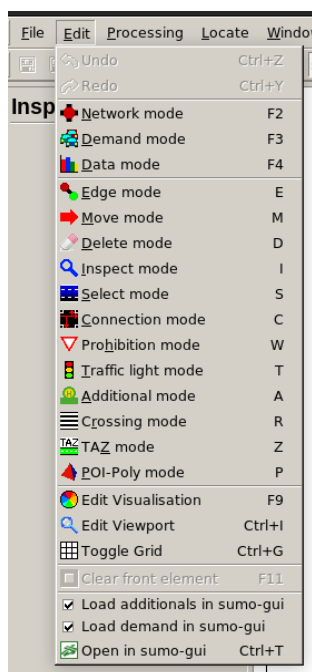
```
duarouter -t file_trips.xml -n file.net.xml -o file.rou.xml
```

*Figura 15: Duarouter eseguito da riga di comando*

Nella sezione successiva 3.3 verranno presentati alcuni tools in Python presenti nel pacchetto Sumo , tra cui randomTrips, che permette di ottenere file\_trips.xml casuali per grandi simulazioni.

### 3.2.3 Configurazione

Una volta ottenuti i file di rete e il file routes, è possibile creare il file di configurazione per effettuare la simulazione. Anche in questa situazione l'interfaccia grafica del pacchetto Sumo è un grosso aiuto. Se infatti, la rete è stata creata tramite netedit, da interfaccia grafica e popolata allo stesso modo, tramite la voce edit del menù di netedit, dopo aver salvato le modifiche alla rete e le entità inserite, è possibile aprire la simulazione con il tasto *open in sumo-gui* :



Così facendo, la simulazione sarà aperta in sumo-gui e basterà avviarla per vedere la simulazione. Come facilmente immaginabile, questa modalità resta comoda per simulazioni semplici, nelle quali i file da utilizzare sono solo file.net.xml e file.rou.xml. Ci sono però situazioni in cui i file da importare sono diversi. In questo caso il file di configurazione deve essere modificato manualmente. Ad esempio è possibile definire file.xml per il funzionamento della segnaletica, oppure per il trasporto pubblico. Non sono file necessari ma rendono più dettagliata la simulazione qualora si voglia.

La struttura base di un file di configurazione è la seguente:

```
<configuration>
  <input>
    <net-file value="Prova.net.xml"/>
    <route-files value="Prova.rou.xml,peds.rou.xml"/>
  </input>
  <time>
    <begin value="0"/>
    <end value="10000"/>
  </time>
</configuration>
```

Figura 16: Contenuto di un file.sumocfg

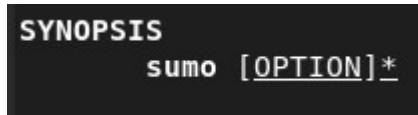
Come è visibile, è possibile modificare i file di input, che ovviamente per quanto riguarda i file di route sono calcolati sulla rete riportata in **<net-file value="....."/>**. Da questo tipo molto semplice è possibile arrivare file di configurazione più precisi come :

```
<input>
  <net-file value="osm.net.xml"/>
  <route-files value="osm.rou.xml,peds.rou.xml"/>
  <additional-files value="osm.poly.xml"/>
</input>
<output>
  <netstate-dump value="base.xml" />
  <!--<full-output value="full_output.xml" />-->
  <fcd-output value="fcd_dump.xml" />
</output>
<processing>
  <ignore-route-errors value="true"/>
</processing>
<routing>
  <device.rerouting.adaptation-steps value="18"/>
  <device.rerouting.adaptation-interval value="10"/>
</routing>
<report>
  <verbose value="true"/>
  <duration-log.statistics value="true"/>
  <no-step-log value="true"/>
</report>
<gui_only>
  <gui-settings-file value="osm.view.xml"/>
</gui_only>
```

Figura 17: Esempio di file.sumocfg più dettagliato

La figura 16 e la figura 17 raffigurano due esempi di *file.sumocfg* ma con dettagli differenti, sono accomunati dalla sezione di input che vedremo meglio in seguito.

Per comprendere al meglio il contenuto del *file.sumocfg*, vediamo in dettaglio le varie opzioni presenti, prima però è bene ricordare che sumo oltre ad essere un pacchetto di simulazione con interfaccia grafica è utilizzabile anche da linea di comando, ed è quindi utilizzabile con diverse opzioni che solitamente andrebbero aggiunte su riga di comando.



```
SYNOPSIS
sumo [OPTION]*
```

Figura 18: Sinossi sumo

Per comodità è possibile creare un *file.sumocfg* con diverse opzioni, ovviamente tra le opzioni non possono mancare *input*, *output*. Vediamo nel dettaglio alcune opzioni:

- Configuration: se per comodità è stato definito un *file.sumocfg* è bene che da riga di comando sia utilizzata l'opzione **-c** per caricare la configurazione all'avvio, ed è intuibile che questo tipo di opzione non può far parte del file stesso;
- input: opzione fondamentale, per caricare i file di rete e di route, e se la simulazione lo prevede, con questa opzione devono essere caricati anche i file aggiuntivi come ad esempio i file per il controllo dei semafori. È possibile utilizzare le opzioni **-n** per caricare i net-file e **-r** per caricare i route-file; oppure se utilizzando un file di configurazione è necessario creare una sezione di input.

```
<input>
  <net-file value="..." />
  <route-files value="..." />
  ....
</input>
```

Oltre i file di rete e di route è possibile anche caricare ulteriori file nella configurazione, come i file addizionali. L'elenco completo è disponibile in Sumo.

- Output: in Sumo sono disponibili anche opzioni per diverse tipologie di output che però saranno visti nel dettaglio successivamente.

- Time: è possibile definire il tempo in secondi, **-b** tempo di inizio, **-e** tempo di fine e d è anche possibile definire un passo **-step-length**. Ovviamente anche questa opzione è possibile riportarla in un file di configurazione come visibile in figura 16.
- Processing: opzione che permette di impostare delle regole durante la simulazione, ad esempio in figura 17 , **<ignore-route-errors value="true" />** non controlla se i percorsi sono collegati , di default è su false. Potrebbe succedere che in alcune particolari modifiche venga cambiato un edge e per non modificare nuovamente il file di rete , se i percorsi sono stati calcolati in precedenza correttamente, si deve ignorare un eventuale errore su quell'edge. É preferibile però effettuare la simulazione su false per non incorrere in errori non comprensibili.
- Routing: opzione che permette di applicare delle regole sul comportamento delle entità della simulazione. Maggiori informazioni sono disponibili in Sumo.
- Report: un'opzione che restituisce , dopo aver eseguito sumo da riga di comando , restituisce un output limitato su statistiche della simulazione e sulle entità . Da non confondere dalle opzioni di output che saranno chiarite successivamente e che sono molto dettagliate. Le opzioni di report riguardano eventuali errori , oppure statistiche sulle entità della simulazione come ad esempio il numero totale.

Queste sono solo alcune delle opzioni che il pacchetto Sumo mette a disposizione, molte altre sono disponibili in Sumo. Come detto in precedenza c'è molta libertà nell'utilizzo delle opzioni, ma è molto importante che i file caricati siano compatibili tra loro , altrimenti c'è il rischio di simulazioni non realistiche o addirittura simulazioni interrotte per colpa di errori di route. Infatti uno dei maggiori errori è quello di utilizzare net-file e route-file non compatibili con bordi e nodi inesistenti per uno dei due file. Quindi è corretto che se un oggetto A si riferisce ad un oggetto B, contenuti entrambi in file differenti, allora il file contenente l'oggetto B deve essere caricato prima del file contenente l'oggetto A.

Inoltre , come detto in precedenza Sumo è un pacchetto Open Source e questo ha permesso sia un'ampia diffusione ma anche la nascita di una comunità attiva, infatti nel pacchetto Sumo oltre le applicazioni viste fino ad ora, sono presenti anche tools che permettono un utilizzo semplificato, come ad esempio nella creazione di net-file.

### 3.3 Tools

I problemi, nonostante il pacchetto Sumo sia ricco di applicazioni come è stato precedentemente detto, possono essere di diverso tipo, ad esempio come poter generare flussi elevati durante le simulazioni. Oltre le applicazioni fornite dal pacchetto Sumo, alcune delle quali viste nelle sezioni precedenti, ci sono molti tools scritti in Python che possono risolvere questi problemi. Ci sono diversi tipi di tools, le tipologie possono riguardare strumenti per la visualizzazione degli output, anche graficamente, oppure simulare il comportamento di una città reale attraverso osmWebWizard, o ancora vedere la differenza tra due net-file, convertire da sumo-output (XML) a un file csv.

Per poter utilizzare correttamente i tools è necessario impostare il percorso ,che per gli utenti di linux :

```
export SUMO_HOME=/your/path/to/sumo-git/
```

*Figura 19: Impostazione temporanea della variabile d'ambiente*

Dopo aver effettuato questo cambiamento da riga di comando, è possibile utilizzare correttamente i tools di sumo. Per la procedura completa e anche per una soluzione permanente, o per configurazioni su altri sistemi operativi vedere in Sumo.

#### 3.3.1 osmWebWizard

Dopo un'attenta analisi e studio, questo script è sicuramente tra i più importanti perché permette di ottenere delle tracce realistiche basate su Open Street Maps, grazie ad una

pagina web che viene aperta. OSM già permetteva una download del file di rete ma con questo tools si può ottenere uno scenario completo. Vengono creati i file dello scenario e salvati in locale per poi essere utilizzati chiamando da riga di comando `sumo -c osm.sumocfg` , oppure direttamente aprendo su `sumo-gui` la configurazione. La chiamata per avviare `osmWebWizard` è ***python tools/osmWebWizard.py*** che richiede che la variabile di ambiente `SUMO_HOME` sia impostata correttamente per funzionare.

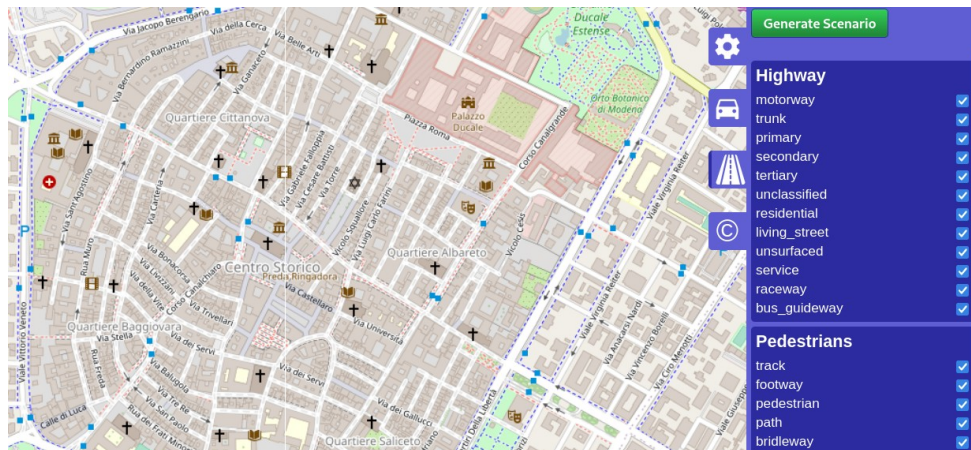


Figura 20: browser web aperto da *osmWebWizard.py*

Quello che viene restituito dopo la chiamata `osmWebWizard` è open street maps con la possibilità ,tramite un menù presente sulla destra di selezionare le strade, tipologie di strade (percorso permesso solo al trasporto urbano ad esempio ) , i pedoni . Tramite il menù è possibile selezionare le città di nostro interesse liberamente. Una volta che sono stati definiti tutti i parametri da selezionare basta cliccare su *Generate Scenario* per avere i nostri file, che si presenteranno in questo modo nella cartella di nostro interesse

<code>base.xml</code>	<code>osm_bbox.osm.xml.gz</code>	<code>osm.passenger.trips.xml</code>	<code>osm.view.xml</code>
<code>build.bat</code>	<code>osm.netccfg</code>	<code>osm.polycfg</code>	<code>peds.rou.xml</code>
<code>fcd_dump.xml</code>	<code>osm.net.xml</code>	<code>osm.poly.xml.gz</code>	<code>run.bat</code>
<code>NUL</code>	<code>osm.net.xml.gz</code>	<code>osm.sumocfg</code>	<code>trips.trips.xml</code>

Figura 21: File ottenuti dallo scenario generato tramite *osmWebWizard.py*

Dopo aver ottenuto la cartella con il timestamp di generazione, basterà avviare `sumo` importando la `osm.sumocfg` da riga di comando o da GUI, come precedentemente detto e far partire la simulazione.

Altri tools sono disponibili per ottenere file di rete da OSM (osmGet.py per ottenere una vasata area da OSM, osmBuild.py per creare un file di rete,etc ) , per maggiori dettagli vedere sulla documentazione di Sumo.

Questi scenari ottenuti tramite questo tools sono anche modificabili , qualora si volesse ad esempio aumentare le entità al suo interno oppure per vedere il comportamento di una singola entità, nel caso di questa tesi le simulazioni sono state fatte soprattutto sull'entità persona senza particolari attenzioni sui comportamenti specifici, ma cercando di simulare scenari reali ( come ad esempio Modena centro ) con un alto tasso di pedoni e vedere gli output anche tramite l'utilizzo di due script. Questa parte però sarà vista nella sezione successiva riguardante gli Output, ma è stato necessario nominarla per introdurre il tool successivo, utile proprio a modificare le entità all'interno di una simulazione e anche i loro comportamenti attraverso di verse opzioni.

### 3.3.2 randomTrips

Questo tool genera una serie di percorsi casuali per una determinata rete che è selezionabile con l'opzione **-n**. Il file in output è in formato XML, se non viene impostato con **-o** un file di output, il file predefinito è trips.trips.xml. La chiamata classica è :

```
python tools/randomTrips.py -n net-file
```

Genera così un file trips.trips.xml con i percorsi in un tempo che se non viene impostato con l'opzione **-e** è di 3600 secondi. Lo script controlla se la destinazione che viene generata è completamente collegata altrimenti non completa alcuni viaggi per evitare errori durante la simulazione , ed è un'attività che viene eseguita dal router. Tra le altre opzioni è possibile impostare la **--min-distance** *valore*. Tra le opzioni sicuramente più importanti c'è **--random** , tra le più importanti perché randomTrips.py se eseguito due volte con gli stessi parametri restituirà gli stessi percorsi , mentre con questa opzione viene richiesta esplicitamente la casualità dei percorsi.

Le entità da utilizzare per il popolamento di una rete devono essere specificate, altrimenti randomTrips.py genera esclusivamente percorsi. Durante lo studio di questo simulatore è diventato ben chiaro che è possibile utilizzare le tipologie di veicoli/persona disponibili tra



le tipologie messe a disposizione dal pacchetto di Sumo , oppure creare un file addizionale contenente una nuova tipologia.

```
<additional>
  <vType id="myType" maxSpeed="27" vClass="passenger"/>
</additional>

python tools/randomTrips.py -n <net-file> --trip-attributes="type=\"myType\"" --additional-file <add-file>
--edge-permission passenger
```

Figura 22: Esempio di un nuovo tipo di veicolo

Fonte: <https://sumo.dlr.de/docs/Tools/Trip.html>

In questo caso , prima si crea il file addizionale e si importa successivamente alla chiamata di randomTrips.py come visibile nella figura 22, altrimenti la classe di default sarà impostata in una simulazione con veicoli non definiti.

Per una simulazione di pedoni invece è sufficiente usare l'opzione **--pedestrians** per generare un simulazione con entità pedoni. Utile per simulare aree pedonali e monitorarne i comportamenti attraverso alcuni script. D'altro canto è evidente che potrebbero essere necessarie simulazioni con comportamenti differenti dei pedoni , e questo è gestibile attraverso altre opzioni di randomTrips.py ma che non sono state analizzate per lo studio inerente a questa tesi. Per maggiori dettagli è quindi necessario consultare la documentazione di Sumo e randomTrips.py. Infine vediamo un file trips.trips.xml che popolerà una simulazione di traffico pedonale, generato dalla chiamata di randomTrips.py:

```
python3 randomTrips.py --n osm.net.xml -r peds.rou.xml --pedestrians -t
"type=\"DEFAULT_PEDTYPE\"" -p 1.4 -e 10000 -l --random --seed 10000
```

```
3624 <person id="1193" type="DEFAULT_PEDTYPE" depart="1670.20">
3625 <walk edges="29241566#0 522431207 28719125#1 522833744 522833741#1 458676577#2 1000553426#1 1000553424 522497449 23195328#0 429515619
429515619#1 429515619#2 1094949085#0 1094949085#1 1094949085#2 23195340#0 192464426"/>
3626 </person>
3627 <person id="1194" type="DEFAULT_PEDTYPE" depart="1671.60">
3628 <walk edges="993930098 997769915 997769913#1 23195339 116985957#1 64973455#1 64973455#0 1094949085#1 1095603363 1095603371#1
1095603370#4 1095603370#1 1095603369#0 1095603369#1 23195325#1 1094949083 191979118#1 656804230 545722933#1 545722931 28719191#1 645997037
645997038 23195329#1"/>
3629 </persons>
```

Figura 23: file trips.trips.xml

### 3.3.3 Altri tool

Altri tool disponibili sono stati citati in precedenza. Ne vediamo alcuni non in dettaglio, ma è giusto parlarne perché interessanti per essere approfonditi:

- Visualization :offerto da Sumo per analizzare gli output di simulazioni con molti dati generati. Gli strumenti sono implementati in python e sfruttano la libreria di matplotlib. Un tool provato è `plotXMLAttributes.py -x x -y y -s -o <nomeGrafico.png> <file_output.xml> --scatterplot`, legge un file di output che generalmente deve essere generato con `-fcd-output` e restituisce un grafico a linee che di solito rappresenta la mappa dei percorsi effettuati dalle entità, i colori non rappresentano l'affluenza delle zone ma i movimenti di alcuni pedoni. Resta valido per simulazioni non enormi;

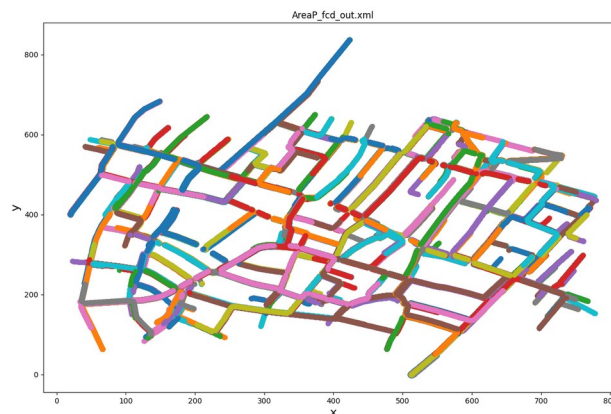


Figura 24: Output di `plotXMLAttribute.py`

- xml: tools molto utili per le simulazioni con file xml molto grandi e difficili da analizzare. Ad esempio il tool `xml2csv.py`, come intuibile dal nome converte un file xml in un file tabellare, un tipo di file molto più comodo da manipolare per delle analisi. Viceversa esiste un altro tool per passare da csv a xml.

Come abbiamo visto i tool sono molteplici e facilitano l'uso di Sumo rendendolo accessibile a tutti i tipi di utenze.

Oltre i tool e le applicazioni, sumo permette anche lo studio dell'output delle simulazioni, fornendo opzioni per la simulazione che forniscono output che variano da output di monitoraggio del traffico, a output sulle emissioni dei veicoli durante la simulazione. Alcuni di questi saranno approfonditi di seguito.

### 3.4 Output

In Sumo, anche gli output sono gestiti direttamente da alcune opzioni. I file che vengono generati sono in formato XML, ma come è stato precedentemente detto non si tratta di un limite data la presenza di tool di conversione in file tabellari più semplici da manipolare per gestire i dati. Le opzioni di output sono di diverso tipo e possono essere utilizzate in due modi :

- riga di comando: come per altre opzioni è possibile utilizzare anche le opzioni di output , ad esempio `sumo -c file.sumocfg --opzione_out <File_out.xml>`. Molto importante selezionare un file di out.
- File di configurazione: un'altra possibilità è quella di inserire l'opzione direttamente nel file di configurazione come visto in 3.2.3. In questo caso, non sarà necessario far altro che eseguire `sumo -c file.sumocfg` oppure caricare la configurazione in `sumo-gui` ed eseguire la simulazione. Al termine della simulazione il `file_out.xml` sarà salvato nella directory con i file di quella simulazione. Ecco un esempio:

```
<output>
  <netstate-dump value="base.xml" />
  <!--<full-output value="full_output.xml" />-->
  <fcd-output value="fcd_dump.xml" />
</output>
```

Figura 25: Esempio definizione output in file di configurazione

In Sumo sono presenti diverse opzioni di output. La prima che sarà approfondita è quella utilizzata successivamente per lo studio effettuato; si tratta di **`-fcd-output`** (floating car data).

Questo tipo di opzione è utile per simulazioni con entità veicolo/persona. L'output si comporta come un GPS super preciso. È anche possibile rendere l'output più selettivo con ulteriori opzioni di `-fcd-output` stesso.

In seguito un esempio per comprendere meglio questo tipo di output, basato su una simulazione effettuata con esclusivamente entità persona:

```
<timestep time="8.00">
  <person id="0" x="215.10" y="448.83" angle="117.38" speed="1.27" pos="4.99" edge="-191979122" slope="0.00"/>
  <person id="1" x="142.38" y="46.85" angle="38.66" speed="1.06" pos="103.32" edge="-322548792#1" slope="0.00"/>
  <person id="2" x="365.38" y="539.83" angle="116.58" speed="1.39" pos="6.46" edge="191871225#1" slope="0.00"/>
  <person id="4" x="680.01" y="422.67" angle="289.84" speed="1.28" pos="2.62" edge="191987468#1" slope="0.00"/>
  <person id="5" x="65.76" y="1.38" angle="344.66" speed="1.08" pos="1.08" edge="28710621#0" slope="0.00"/>
  <person id="3" x="266.07" y="225.75" angle="43.99" speed="1.31" pos="4.16" edge="322548792#6" slope="0.00"/>
</timestep>
<timestep time="9.00">
  <person id="0" x="216.19" y="448.26" angle="117.38" speed="1.23" pos="6.22" edge="-191979122" slope="0.00"/>
  <person id="1" x="143.19" y="47.86" angle="38.66" speed="1.29" pos="102.03" edge="-322548792#1" slope="0.00"/>
  <person id="2" x="366.41" y="539.32" angle="116.58" speed="1.15" pos="7.62" edge="191871225#1" slope="0.00"/>
  <person id="4" x="678.77" y="423.12" angle="289.84" speed="1.32" pos="3.94" edge="191987468#1" slope="0.00"/>
  <person id="5" x="65.49" y="2.38" angle="344.66" speed="1.03" pos="2.11" edge="28710621#0" slope="0.00"/>
  <person id="3" x="267.07" y="226.78" angle="43.99" speed="1.43" pos="5.59" edge="322548792#6" slope="0.00"/>
  <person id="6" x="224.15" y="194.35" angle="45.40" speed="0.00" pos="0.00" edge="997944687" slope="0.00"/>
</timestep>
```

Figura 26: Rappresentazione in formato xml di un output di floating car data (fcd-output)

Come è possibile vedere dalla figura 26 il contenuto del file è diviso in timestep, che di default è 3600 secondi. Il passo temporale se non definito è di un secondo. Per ogni timestep, in uno scenario di pedoni, i dati che fcd-output restituisce riguardano:

- **id:** identificativo di ogni persona;
- (x,y): coordinate della persona intese come latitudine e longitudine;
- **angle:** l'angolo di navigazione (0-360°);
- **speed:** velocità intesa al m/s;
- **pos:** spostamento dall'inizio della simulazione;
- **edge:** il bordo sul quale si trova l'entità;
- **slope:** pendenza;

La simulazione non comprende solo queste 6 entità, ma come già definito, nella creazione della route con il tool randomTrips.py, inserendo l'opzione `--random-depart` si sta esplicitamente dicendo di dare i tempi di inserimento ad ogni entità casuali, infatti sempre nella figura 26, se si presta attenzione si può notare che tra il timestep 8 e il timestep 9, **person id=6** è appena entrata in simulazione, tant'è che il valore di **pos=0**, quindi non ha effettuato ancora uno spostamento. In sumo oltre questa opzione standard, ci sono opzioni più selettive come ad esempio `--fcd-output.signals` che in caso di presenza di

veicoli restituisce informazioni sui segnali luminosi dei veicoli stessi.

Altri tipi di output sono –emission-output. L'utilizzo è identico alle altre opzioni di output, quindi necessita di un file in formato xml dove salvare i dati generati. Questa opzione genera dati sulle emissioni dei veicoli per ogni timestep della simulazione. I dati delle emissioni riguardano Co2,NOX, carburante utilizzato, rumore e altre informazioni di questo genere.

E ancora , è possibile utilizzare opzioni per avere dati sui bordi, sulla segnaletica,sui veicoli , sulla rete sui semafori. Per alcune di queste informazioni esiste anche una opzione che le racchiude tutte ma non utilizzata spesso per via della grande quantità di dati che genera, alcuni dei quali superflui. Si tratta dell'opzione full-output che contiene info su veicoli/persone,semafori, linee e bordi della rete. Il file xml conterrà , per ogni passo di timestep tutte queste informazioni. Maggiori dettagli sugli output disponibili sono ottenibili in Sumo.

### **3.4.1 tools aggiuntivi di analisi output**

Per il completamento del capitolo è necessario si parlerà di due tools per l'analisi degli output generati con l'opzione fcd-output.

I due script , permettono l'analisi del traffico pedonale, ma potrebbero essere adattati a qualsiasi tipo di traffico. Scritti in Python sfruttano diverse librerie , tra cui Pandas per salvare i dati di output in file tabellari, oppure libreria statistics per il calcolo delle deviazioni standard di ogni cella . Il primo è denominato *xmlAnalyzer.py* , analizza un file di output generato da una simulazione su traccia reale , come fatto su un net-file di Modena, popolato con randomTrips.py con una quantità di pedoni superiore al migliaio. Il suddetto script dopo aver preso in input il *file\_dump.xml* , i dati estrapolati da questo file riguardano l'id della persona e le coordinate. Calcola una cella quadrata di dimensione fissa in base alle coordinate associate ad una persona, e questa cella è utilizzata come tupla. Questo script calcola la variazione di posizione di ogni persona, restituendo quante celle ha

visitato, in base a quelle calcolate e queste variazioni vengono poi salvate in un file tabellare.

L'altro script è `xmlGridStats.py` , a differenza dell'altro in questo caso l'obiettivo era quello valutare , definita una griglia, la variazione all'interno di ogni cella. Quindi definita una griglia, viene preso in input il generato da `fcd-output` , e sullo stesso ragionamento precedente è calcolata la posizione di ogni entità persona in base alle sue coordinate in quel momento nella simulazione. Definita la posizione , questa viene associata ad una determinata cella; così facendo ad ogni cella verrà associato un id. Alla fine dell'esecuzione ogni cella restituirà il numero di persone transitate al suo interno, e deviazioni standard. L'output anche in questo caso viene salvato in un file.csv.

Un ultimo script sviluppato sull'idea di un tool di Sumo è `xmlAnalyzer_graphic.py` , che preso in input un `file_output.xml` genera un'immagine della mappa sulla quale è stata effettuata la simulazione, ricavandola da una griglia le cui celle sono colorate. I colori delle celle variano in base al numero di attraversamenti.

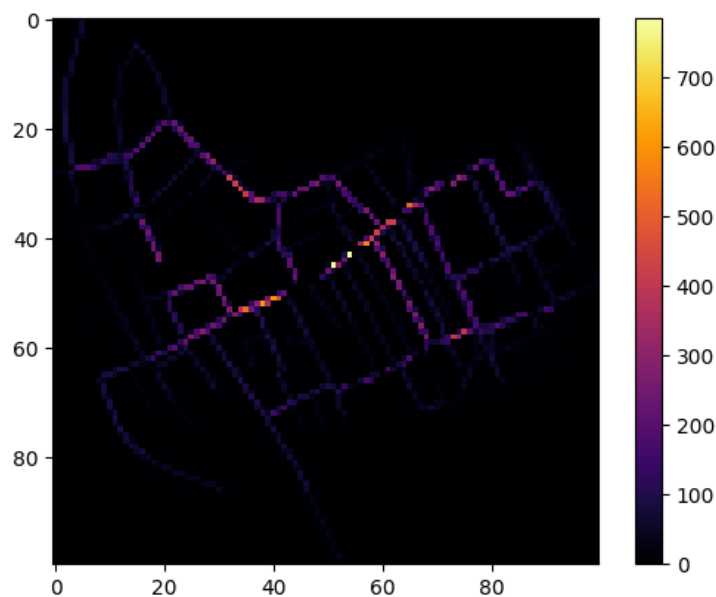


Figura 27: Immagine generata da `xmlAnalyzer_graphic.py`

### 3.4.2 Analisi dei tool su scenario di prova

Per valutare la correttezza dei nuovi tools è stato necessario generare inizialmente uno scenario semplice, definito come scenario di prova.

Per prima cosa, trattandosi di uno scenario semplice, il net-file è stato generato tramite netedit:



*Figura 28: Rete di prova visibile nella GUI di netedit*

Questa rete, composta se vogliamo in parole semplici da due strade, è stata popolata sempre tramite l'interfaccia grafica di netedit, con un singolo pedone che deve effettuare questo percorso, entrando da sinistra uscendo sulla destra. Questo semplice scenario è stato perfetto per poter confrontare un set di output attesi con quelli restituiti dai tools `xmlAnalyzer.py` e `xmlGridstats.py`. Definito un tempo di simulazione pari a 100 secondi, e definito la tipologia di output, ovvero ***-fcd-output***, i due tools produrranno due file.csv differenti

#### **XmlAnalyzer.py**

Già spiegato in precedenza, questo è un tool sviluppato in Python, nello specifico una volta ottenuto un file di output generato tramite l'opzione di Sumo ***-fcd-output***, questo file viene preso in input dal tool in questo modo:

```
$ python3 xmlAnalyzer.py fcd_out.xml
```

*Figura 29: chiamata di xmlAnalyzer.py*

Dal file in input che ha il formato già ,rappresentato in figura 26, vengono estrapolate due tipologie di informazioni , id e coordinate (x,y). Su questi dati , nello specifico viene calcolata la posizione dell'entità e associata ad una cella di una griglia calcolata al momento con dimensione fissa. Dopo aver associato ad ogni entità le celle visitate , queste informazioni vengono salvate su un file.csv che si presenterà in questo modo:

	A	B	C
1	id	celle_visit	num_cambi_cella
2	person_0	{{(10.0, 50.0, 20.0, 60.0), (40.0, 50.0}}	54
3			

Figura 30: Rappresentazione output di xmlAnalyzer.py

Nella figura 30 è rappresentato l'output del tool, colonna per gli id dove vengono rappresentate tutti gli id delle entità, una colonna per le celle visitate, nel quale le celle sono rappresentate da quattro valori (x,y,x+cell\_size,y+cell\_size; che rappresentano il punto in alto a destra della cella e il punto in basso a sinistra) e un contatore per il numero di cambio cella che mantiene un conteggio sulle nuove celle visitate non contando quelle già visitate.

### XmlGridstat.py

L'altro tool sviluppato in Python , è simile al precedente , permette di monitorare le celle di una griglia in ambito di mobilità urbana. Utilizzato in una simulazione permette quindi di avere informazioni sulla quantità di entità che hanno transitato in una determinata cella della griglia. Come per il tool precedente, ha bisogno di un file\_output.xml preferibilmente generato tramite l'opzione fcd-output per ricavare le stesse informazioni, id e coordinate(x,y). A differenza del precedente, con l'utilizzo di questo tool la griglia viene costruita prima del calcolo delle posizioni di ogni entità e ogni cella viene salvata in un dizionario. Solo dopo vengono calcolate le posizioni delle varie entità ed associate ad una cella. La chiamata è così effettuata:



```
$ python3 xmlGridstats.py fcd_out.xml
```

Figura 31: Chiamata del tool xmlGridstats.py

Alla fine della chiamata , l'output è salvato in un file.csv strutturato in questo modo:

cell	num_people	std_dev_x	std_dev_y
(-50, -50, -40, -40)	0	0	0
(-50, -40, -40, -30)	0	0	0
(-50, -30, -40, -20)	0	0	0
(-50, -20, -40, -10)	0	0	0
(-50, -10, -40, 0)	0	0	0

Figura 32: Rappresentazione output xmlGridstats.py

Nella figura 32 è possibile osservare la struttura dell'output generato, che presenta una colonna per le celle , dove ad ogni riga , quindi ad ogni cella identificata da quattro valori (x,y, x+cell\_size,y+cell\_size) ,quattro valori che identificano due punti( punto in alto a destra e punto in basso a sinistra della cella ), è associato il numero di persone transitate e le deviazioni standard di ogni cella , intesi come la variazione dei punti in una cella per valutare le zone più “dense” della cella stessa.

Lo sviluppo di questi due tools è una base, e sicuramente possono essere apportate modifiche future come il filtraggio dei dati di output in base al timestep. Inoltre utilizzarli entrambi per la stessa simulazione, in simulazioni semplici al momento, permette di effettuare anche un controllo incrociato perché i dati saranno gli stessi scritti però in maniera differente. Ad esempio, nella simulazione di prova , avendo un singolo *person\_0* e sapendo a quale cella dovesse essere associato, è bastato controllare i file di output per ritrovare i dati salvati correttamente per entrambi gli output dei due script. I due tools sono una base di partenza, un'ulteriore modifica potrebbe prevedere anche una lista di id per ogni cella in xmlGridstats.py in modo da poter confrontare gli id con quelli dell'output di xmlAnalyzer.py per verificarne la correttezza per la simulazione sullo stesso scenario.

Infine, tornando alla rappresentazione della rete e la costruzione di questa griglia è chiaro come la griglia non sia realmente rappresentata sulla rete ma allo stesso modo sfrutta i movimenti delle entità durante le simulazioni come se lo fosse, sfruttando la i dati che si

hanno sulla rete, che al momento della creazione di una traccia vengono assegnate a quella traccia, ma questo passaggio si comprenderà meglio di seguito, nella presentazione di una simulazione completa di uno scenario reale.

### 3.5 Simulazione completa di uno scenario reale

In conclusione del capitolo 3 , sarà mostrata una simulazione completa di mobilità urbana in ambito pedonale, dalla creazione della traccia fino ad arrivare all'analisi finale effettuata con i tools sviluppati .

#### 3.5.1 Creazione net-file

Lo scenario che si andrà a presentare riporta la zona di Modena centro. Il file.net.xml , è stato ottenuto tramite il tool osmWebWizard:

*python osmWebWizard.py .*

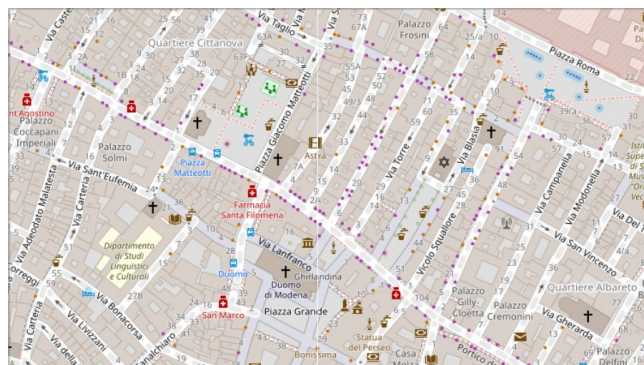


Figura 33: Contenuto del file *osm.net.xml* visualizzato su *netedit*

Nella figura 33 è possibile osservare come si presenta una rete sulla GUI di Netedit e di Sumo , rispetto a come si presenta su Open Street Maps. Viene effettuata una proiezione della mappa sul piano cartesiano , che in questo caso è racchiusa nell'area tra i punti  $(x1,y1)=(0,0)$  e  $(x2,y2)=(780.94,773.83)$  come riportato anche nel file *osm.net.xml* all'elemento *location* nell'attributo *convBoundary*,

Da Open street maps sono stati ricavate anche le entità contenute in `osm.net.xml` , che però saranno sostituite durante la simulazione , e anche le regole per le strade, sempre contenute nel `file.net.xml`

### 3.5.2 Creazione del file route

La simulazione fornita dal tool `osmWebWizard.py` è già completa di ogni file necessario, e anche del file di route. Però impone limiti su simulazioni reali con grandi numeri di entità, per questo motivo per delle simulazioni più dettagliate, o specifiche per un determinato tipo di entità è necessario o modificare il file di route oppure generarne uno nuovo.

Per svolgere una simulazione sulla traccia di Modena, simulando il comportamento di una grossa quantità di pedoni si è utilizzato il tool `randomTrips.py` nel seguente modo:

```
python3 randomTrips.py --n osm.net.xml -r peds.rou.xml --pedestrians -t  
"type='DEFAULT_PEDTYPE'" -p 1.4 -e 1000 --random-depart --random --seed 10000
```

Analizzando la chiamata:

- **-n:** il file di rete contenente i dati sui bordi, nodi ,etc sui quali saranno generati i percorsi;
- **-r:** il file sul quale andremo a salvare i percorsi, di default è `trips.trips.xml`;
- **--pedestrians:** opzione che genera traffico pedonale;
- **-t:** definiamo i pedoni di tipo `default_pedtype`, presente su `netedit`;
- **-p:** periodo di inserimento dei pedoni nella simulazione, in secondi;
- **-e:** ora di fine in secondi, di default 3600;
- **--random-depart:** inserimento randomico nella simulazione, senza una posizione definita;

- **--random**: opzione consigliata per rendere ogni generazione di percorsi casuale;
- **--seed**: utile per ottenere una ripetibilità della simulazione effettuata, usando gli stessi dati e lo stesso intero.

Il file di route ottenuto conta più di 700 pedoni. Ora bisogna modificare il file di configurazione ottenuto tramite osmWebWizard.

### 3.5.4 Modifica file di configurazione e simulazione

Per effettuare la simulazione l'ultimo passo è quello di modificare il file di configurazione `osm.sumocfg`. Come è stato detto nelle sezioni precedenti il file di configurazione può contenere diverse opzioni accettabili da Sumo. In questo caso ci basterà inserire il file `peds.rou.xml` nell'attributo **route-files** in `osm.sumocfg`:

```
<input>
  <net-file value="osm.net.xml.gz"/>
  <route-files value="peds.rou.xml"/>
  <additional-files value="osm.poly.xml.gz"/>
</input>
```

Figura 34: Elemento input del file `osm.sumocfg`

Modificando solo questa parte, non ci resta che eseguire la simulazione a piacere o da GUI aprendo il file `osm.sumocfg` tramite `sumo-gui`. Oppure, nel caso si volesse generare anche un output, direttamente da riga di comando: `sumo -c osm.sumocfg <opzione_output> <file_out.xml>`. Inoltre è corretto ricordare che l'elemento output e l'opzione specifica per un determinato tipo di output può essere definita direttamente nel file di configurazione, come d'altronde è stato fatto in questa simulazione.

```
<output>
  <netstate-dump value="base.xml" />
  <!--<full-output value="full_output.xml" />-->
  <fcd-output value="fcd_dump.xml" />
</output>
```

Figura 35: Elemento output nel file di configurazione

### 3.5.5 Output e analisi dei dati ottenuti

Dopo aver effettuato la chiamata di sumo sul file di configurazione , contenente l'opzione di output desiderata, in questo caso di `-fcd-output`, verrà generato un file contenente i dati di ogni persona e dei percorsi effettuati.

```
<person id="521" x="395.44" y="525.56" angle="114.19" speed="1.29" pos="11.69" edge="191871225#2" slope="0.00"/>
<person id="526" x="161.76" y="436.17" angle="21.38" speed="1.42" pos="31.61" edge="191959704#1" slope="0.00"/>
<person id="365" x="159.22" y="482.85" angle="116.88" speed="1.31" pos="13.29" edge="191959714" slope="0.00"/>
<person id="259" x="256.00" y="433.61" angle="295.41" speed="0.94" pos="7.27" edge="191979121" slope="0.00"/>
<person id="440" x="246.29" y="438.93" angle="295.41" speed="1.34" pos="18.33" edge="191979121" slope="0.00"/>
<person id="473" x="212.30" y="381.64" angle="114.54" speed="1.27" pos="30.58" edge="191979127#0" slope="0.00"/>
<person id="210" x="203.34" y="386.10" angle="287.33" speed="1.26" pos="40.56" edge="191979127#0" slope="0.00"/>
<person id="477" x="159.81" y="397.67" angle="107.32" speed="1.22" pos="12.66" edge="191979127#3" slope="0.00"/>
<person id="432" x="561.23" y="238.52" angle="117.76" speed="1.16" pos="2.00" edge="191987465#0" slope="0.00"/>
```

Figura 36: Frammento dell'output generato

Questi dati ai fini di un'analisi approfondita, possono essere gestiti con i due tools sviluppati `xmlGridStats.py` e `xmlAnalyzer.py` .

#### XmlGridStats.py

Una volta generato il file output, come vediamo in figura 36, possiamo effettuare la prima analisi attraverso `xmlGridStats.py`, andando così ad effettuare una verifica sullo stato delle celle. Effettuo la chiamata :

```
python3 path/to/xmlGridStats.py path/to/file_output.xml
```

Il file generato in formato CSV, è salvato nella directory in cui viene eseguito lo script. Presenterà 4 colonne : `id_cella`, numero di persone transitate e le deviazioni standard.

(130, 450, 140, 460)	2.0	3.259762261269994	6.7104433534603025
(130, 460, 140, 470)	2.0	0.84145706961199	3.9173715677734475

Figura 37: Frammento dell'output di `xmlGridStats.py`

Nella figura 37 è possibile vedere una parte del file CSV generato, ad esempio la prima riga la cella  $((x1,y1),(x2,y2))=(130,450,140,460)$  è stata attraversata da 2 persone durante la simulazione, con una deviazione standard delle  $x = 3,25$  e una deviazione standard delle  $y=6.7$ .

Da ricordare, come viene proiettata la mappa, spiegato nella sezione 3.5.1.

#### xmlAnalyzer.py

L'altro tool utile per un'analisi più approfondita è xmlAnalyzer.py. La chiamata da effettuare è:

***python3 path/to/xmlAnalyzer.py path/to/file\_output.xml***

Il file generato è in formato CSV, ed è composto da tre colonne , id ,celle\_visitate\_numero cambi cella.

id	celle_v	num_cambi_cella
0	{(140.0	84
1	{(470.0	80
2	{(260.0	90
3	{(230.0	70

*Figura 38: Frammento dell'output generato da xmlAnalyzer.py*

Nella figura 38 è possibile osservare il contenuto dall'analisi dell'output generato dalla simulazione di Modena, ad esempio nella prima riga vediamo che la persona con **id 0** ,durante la simulazione (che ricordo essere di 10000 secondi) ha effettuato 84 **cambi di celle** , visitando la lista di celle della colonna **celle\_visit**. Da ricordare, inoltre, che un cambio cella viene conteggiato se questo cambio avviene verso celle non visitate, altrimenti resta invariato.

Una volta ottenuta questa quantità di dati , bisogna analizzarli e una possibile analisi è quella di vedere la distribuzione dei pedoni e delle celle .

Analizzando i dati derivati da xmlAnalyzer.py , visibili in parte nella figura 38, è stato possibile generare questo tipo di istogramma.

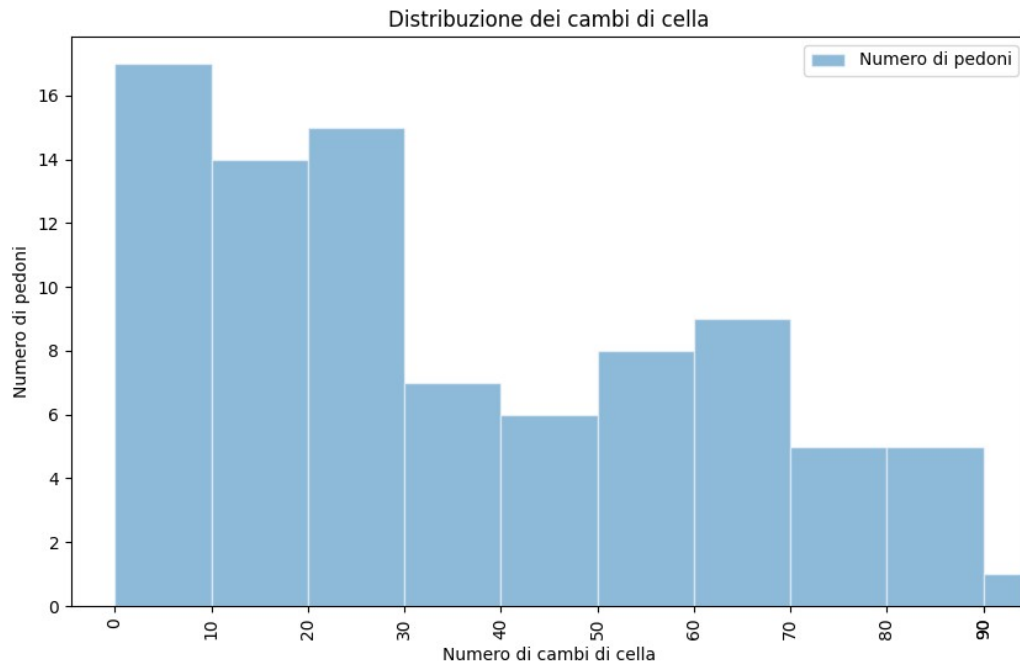


Figura 39: Analisi dei dati di *xmlAnalyzer.py*

Sull'asse x sono rappresentati gli intervalli di cambio cella, mentre sull'asse y quanti pedoni hanno effettuato quei cambi .

Per quanto riguarda l'output di *xmlGridStats.py* , è stato utilizzato lo stesso tipo di analisi , andando a valutare su intervalli di attraversamento il numero di celle corrispondenti.

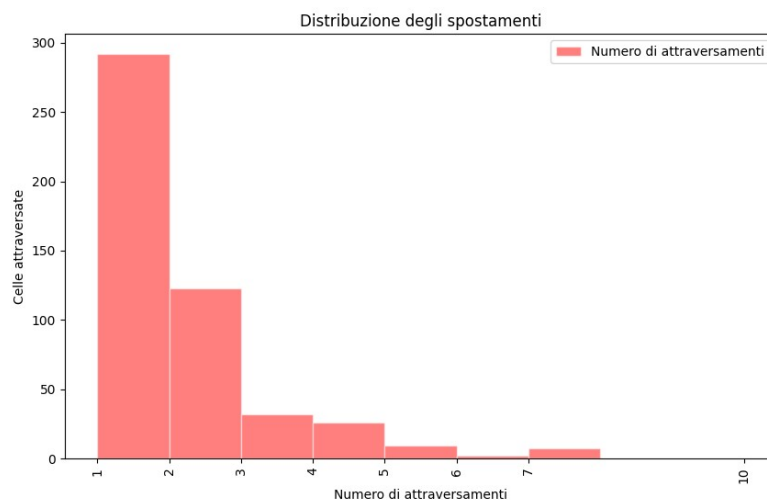


Figura 40: Analisi dell'output di *xmlGridStats.py*

In questo caso , sull'asse x è stato definito il numero di spostamenti effettuati , in base ad un massimo nell'output, mentre sull'asse y il numero di celle attraversate.



## Conclusioni

Lo sviluppo di Internet ha portato alla tecnologia Cloud. Una tecnologia che ha permesso e incrementato lo sviluppo informatico di molte realtà. Molte piccole aziende hanno potuto usufruire di sistemi e servizi da remoto che fisicamente non avrebbero mai potuto possedere per via dei costi elevati di gestione e manutenzione. Come per ogni campo, il continuo sviluppo ha portato alla nascita di progetti ambiziosi difficilmente sviluppabili e sostenibili attraverso il solo paradigma Cloud.

In questa tesi l'attenzione si concentra sullo sviluppo di Smart City , sullo sviluppo della mobilità urbana e ,dal punto di vista teorico, sull' Edge/Fog computing. Si è voluto dare importanza alla mobilità urbana che condiziona valutazioni positive o negative sulle città , valutazioni sulla loro gestione, determina le abitudini dei cittadini e ,che come detto all'interno della tesi, non è considerata attivamente nello sviluppo di una città intelligente.

Per permettere una corretta evoluzione e realizzazione di questi progetti , è necessario sì studiare le infrastrutture che permettono le comunicazioni con i veicoli o con altre entità, ma anche il poter simulare il loro funzionamento che , dati i costi , difficilmente possono essere svolti in vere città ed è per questo che il punto centrale di questa tesi è stato il pacchetto di simulazione Sumo, che permette di svolgere simulazioni realistiche di scenari generati da reali informazioni.

Le analisi viste e lo studio fatto in questa tesi forniscono un punto di partenza per lo studio di questo simulatore, in modo da avere chiaro il suo potenziale e inoltre tramite le simulazioni effettuate e lo sviluppo di due tools, vengono fornite delle buone basi di partenza per l'analisi approfondita del comportamento di veicoli/pedoni nell'ambito della mobilità urbana. Inoltre è possibile anche osservare una possibile tipologia di analisi dei dati , ovvero quello sulla distribuzione dei pedoni e degli attraversamenti.

## Bibliografia

- <https://www.gbnrtc.org/smartmobility>
- [https://www.researchgate.net/figure/A-smart-city-architecture-abstract-view-From-the-technical-point-of-view-the\\_fig1\\_326874508](https://www.researchgate.net/figure/A-smart-city-architecture-abstract-view-From-the-technical-point-of-view-the_fig1_326874508)
- <https://www.spiceworks.com/tech/cloud/articles/what-is-cloud-computing/>
- <https://www.spiceworks.com/tech/cloud/articles/what-is-cloudcomputing/>
- <https://vitolavecchia.altervista.org/caratteristiche-e-differenza-tra-cloud-fog-e-edge-computing/>
- <https://www.mdpi.com/2504-2289/2/2/10>
- <https://www.mdpi.com/2504-2289/2/2/10>
- <https://www.mdpi.com/1424-8220/19/5/1073#>
- [https://www.researchgate.net/figure/Conversion-of-network-file-using-Netconvert\\_fig4\\_368943366/download](https://www.researchgate.net/figure/Conversion-of-network-file-using-Netconvert_fig4_368943366/download)
- <https://sumo.dlr.de/docs/Tools/Trip.html>
- <https://sumo.dlr.de/docs/>