POLITECNICO DI MILANO

Facoltà di Ingegneria dell'Informazione

Corso di Laurea Specialistica in Ingegneria Informatica



# Algoritmi per la ricerca di politiche ottime per il routing a due-hop nelle Delay Tolerant Network multi-classe con beaconing


Relatore:

**Prof. Nicola Gatti**

Correlatore:

**Prof. Nicola Basilico**

Tesi di Laurea di:

**Luca Goffredi**

**Matricola 750734**


**Anno Accademico 2012-2013**

# Indice

# Elenco delle figure

# Elenco delle tabelle

# Elenco degli algoritmi

# Sommario

Nelle Delay Tolerant Networks (DTN), il routing di un pacchetto a due hop ha dimostrato di saper trovare un compromesso tra energia e ritardo più conveniente rispetto all'instradamento epidemico. La letteratura fornisce risultati completi sulle politiche di instradamento ottime per nodi mobili con profili di mobilità omogenei, spesso trascurando i costi di segnalazione. Le politiche di routing vengono abitualmente calcolate mediante tecniche di approssimazione fluide, i cui risultati di ottimalità valgono solo in casi teorici in cui il numero di nodi tende ad infinito, mentre forniscono una approssimazione grossolana negli altri casi. In questa tesi si considerano modelli di mobilità eterogenei e molteplici tecnologie di trasmissione wireless, inoltre si considerano esplicitamente i costi di beaconing/segnalazione per supportare il routing e la possibilità per i nodi di scartare i pacchetti ricevuti dopo un tempo prefissato. Abbiamo caratterizzato in modo teorico le politiche ottime derivando le loro proprietà formali. Tale analisi viene quindi sfruttata per definire due approcci algoritmici che permettono di trovare un compromesso tra ottimalità della soluzione e efficienza computazionale. Vengono poi derivati dei bound teorici sulle garanzie di approssimazione degli algoritmi proposti. Abbiamo quindi valutato sperimentalmente i nostri algoritmi in scenari realistici di DTN nel caso multi classe per confermare la validità del nostro approccio.

# Capitolo 1

# Introduzione

Le Delay Tolerant Networks (DTNs) (note anche come Disruptive Tolerant Netwoks) sono reti wireless ad hoc ad alta mobilità e/o deboli che non garantiscono una connettività continua. Alcuni esempi sono le reti wireless ad alta mobilità particolarmente utilizzate in ambito militare in ambienti estremi, oppure reti interplanetarie. Il disturbo può verificarsi a causa dei limiti del range di trasmissione wireless, della dispersione dei nodi mobili, delle risorse energetiche limitate, attacchi esterni e rumore ambientale. Un problema centrale nelle DTN è l'instradamento dei pacchetti da una sorgente verso le destinazioni desiderate. Quando non sono disponibili a priori informazioni sul modello di mobilità dei nodi, una tecnica comune per superare la mancanza di connettività è diffondere più copie del pacchetto nella rete: questo aumenta la probabilità che almeno uno di loro raggiunga il nodo di destinazione entro una determinata deadline temporale. Questo metodo è conosciuto come instradamento epidemico, perché, allo stesso modo nella diffusione di malattie infettive, ogni volta che un nodo che possiede già il pacchetto incontra un nuovo nodo che non possiede una copia dello stesso, il portatore può infettare questo nuovo nodo passandogli una copia del pacchetto; i nodi appena infettati, a loro volta, possono comportarsi allo stesso modo. La destinazione riceve il pacchetto quando incontra un nodo infetto. Tuttavia, l'instradamento dei pacchetti

in modo epidemico porta ad un alto consumo in termini energetici, mentre un compromesso tra ritardo di consegna del pacchetto e costi energetici è fornito dal routing a due hop [17] nel quale solo la sorgente del pacchetto può infettare gli altri nodi i quali spediranno l'informazione solamente alla destinazione corretta. Per questo motivo, per il resto dell'elaborato ci concentreremo su questa tecnica di routing.
Nell'articolo su cui ci siamo basati [7], gli autori studiano le politiche ottime statiche e dinamiche (che si dimostrano essere a soglia) per le DTN a due hop nel caso in cui i nodi mobili appartengono ad una singola classe. In questo caso la politica ottima può essere trovata in forma chiusa. Inoltre, gli autori mostrano che quando i parametri del sistema sono sconosciuti, è ancora possibile ottenere una politica che converge a quella ottima utilizzando alcuni meccanismi adattivi che permettono di ricavare una stima di tali parametri. Un'ulteriore estensione di tale meccanismo adattivo è fornito in [19]. Lo scenario in cui sono presenti più classi di nodi mobili è studiato in [28] in cui si sfruttano alcune approssimazioni fluide e tecniche di controllo ottimo sotto l'assunzione che il numero di copie del pacchetto presenti nella rete aumenta monotonicamente col passare del tempo. Viene dimostrato che le approssimazioni fluide sono esatte quando il numero di nodi tende all'infinito, ma risultano essere grossolane negli altri casi come mostrato, per esempio, in altri campi in [29, 30] dove l'approssimazione fluida è soddisfacente quando il numero di utenti è superiore a 200. In [13], gli autori forniscono la struttura in forma chiusa delle politiche di spedizione del pacchetto e forniscono un algoritmo stocastico per l'apprendimento dei parametri. In [5] vengono proposti dei timer di validità dei pacchetti da associare al messaggio quando viene salvato nel nodo mobile. In [31] e in [11], gli autori ottimizzano le prestazioni della rete pianificando la mobilità dei nodi che trasmettono i messaggi. La teoria dei giochi evolutivi è stata adottata in [10] per incentivare i nodi mobili a prendere il pacchetto e spedirlo alla destinazione desiderata.
Sono stati affrontati anche numerosi problemi correlati a quello principale.

In [9] gli autori affrontano il problema di trasferire file di grandi dimensioni dalla sorgente alla destinazione quando non tutti i pacchetti sono disponibili alla sorgente prima dell'inizio della trasmissione. In [8], viene fornita una classe di meccanismi di replicazione per i pacchetti nella rete che include una codifica per migliorare la probabilità di consegna con successo entro un dato tempo limite. In [15], vengono presi in considerazione nodi selfish/non cooperativi e il problema è quello di incentivare i nodi mobili a spedire il pacchetto. In [14], gli autori si focalizzano su scenari con routing multi hop con l'obiettivo di progettare degli incentivi per i nodi mobili in modo da evitare che questi nodi si comportino strategicamente in termini di edge-hiding e edge-insertion. Un approccio tecnologico simile è proposto in [33]. In [20], gli autori studiano differenti strategie di routing nel caso in cui i nodi possiedono un buffer limitato e forniscono un confronto sperimentale delle strategie. Un lavoro simile è presentato in [24], in cui però gli autori inseriscono delle informazioni sociali che caratterizzano i nodi. Alcuni aspetti cooperativi sono trattati in [26] e in [27].

La principale novità di questo lavoro consiste nell'affrontare uno scenario multi classe in cui vengono considerati anche i costi di beaconing e nel fornire degli algoritmi per trovare le politiche di trasmissione ottime e approssimate nel caso in cui il numero di nodi non è infinito, vale a dire nel caso in cui l'approccio mediante le approssimazioni fluide mostrate nella letteratura mostra i propri limiti. Per fare ciò abbiamo esteso il modello più espressivo di routing a due hop nelle DTN [3, 5, 28] con i seguenti contributi:

- includiamo i costi di beaconing, differenti per ogni specifica tecnologia di trasmissione, consentendo alle classi di nodi di condividere la stessa tecnologia di trasmissione e quindi anche le stesse attività di beaconing;

- permettiamo che il numero di copie del pacchetto non sia monotono crescente nel tempo, assumendo che i nodi scartino il pacchetto dopo una deadline locale;

- studiamo, per la prima volta, algoritmi per la ricerca della soluzione ottima, in tempo esponenziale, e approssimata con un bound teorico, in tempo polinomiale, quando il numero di nodi è finito;

- valutiamo sperimentalmente in modo accurato gli algoritmi proposti con parametri realistici in termini di approssimazione e scalabilità, valutiamo inoltre come le politiche ottime cambiano al variare dei parametri.

La tesi è strutturata nel modo seguente. Nel Capitolo 2 viene riassunto lo stato dell'arte per i problemi di instradamento di pacchetti nelle DTN e di problemi correlati al nostro lavoro anche se per aspetti diversi dall'instradamento.

Nel Capitolo 3 viene introdotto nel dettaglio lo scenario e il modello formale del problema che abbiamo affrontato e vengono fornite alcune proprietà teoriche che saranno sfruttate per la formulazione degli algoritmi.

Nel Capitolo 4 vengono forniti due algoritmi con diversa complessità che permettono di approssimare la politica ottima del problema e vengono inoltre forniti i bound teorici sulla qualità della soluzione trovata.

Nel Capitolo 5, dopo aver fornito i dettagli per i parametri usati negli esperimenti, si confrontano le prestazioni dei nostri algoritmi rispetto ad altri semplici algoritmi euristici facilmente calcolabili e si valuta la qualità della soluzione rispetto ad un upper bound.

Nel Capitolo 6 si fanno alcune considerazioni conclusive sul lavoro svolto, proponendo alcuni possibili sviluppi futuri.

Infine nell'Appendice A si riporta il codice Matlab dei diversi algoritmi che è stato utilizzato per effettuare gli esperimenti.

# Capitolo 2

# Stato dell'arte

Nella prima parte di questo capitolo si vuole dare una visione generale dei problemi affrontati in letteratura, mentre nella seconda parte vengono forniti i dettagli di alcuni dei lavori più interessanti che sono in relazione con il nostro lavoro. Infine si riassumono le caratteristiche comuni ai principali lavori presi in considerazione.

## 2.1 Classificazione dei problemi

Un certo numero di articoli affronta il problema di studiare strategie di spedizione di pacchetti per le Delay Tolerant Networks.
Il modello di base è il seguente: ci sono uno o più pacchetti che devono essere spediti da una sorgente verso una destinazione predefinita. Ogni pacchetto è associato ad una deadline temporale, oltre la quale il pacchetto perde il proprio valore (ad esempio una informazione non più rilevante dopo un certo periodo di tempo). La sorgente e ciascun nodo mobile può avere una strategia di spedizione dei pacchetti potenzialmente diversa.

I principali problemi studiati in letteratura sono i seguenti.

*Problema 1 (problema del nodo sorgente):* Il problema è quello di determinare la strategia ottimale di trasmissione della sorgente con lo scopo di massimizzare la probabilità di consegnare il pacchetto entro il termi-

ne temporale stabilito considerando alcuni vincoli energetici. Un aspetto cruciale è lo studio di strategie che dipendono da numerosi parametri (ad esempio tempo, classe di mobilità dei nodi mobili, classe di comunità dei nodi mobili) e l'analisi del costo di tali parametri (ad esempio, costi di beaconing costi e costo di trasmissione).
Questo problema è abitualmente modellato come un problema di ottimizzazione.

*Problema 2 (problema dei nodi mobili):* Il problema è quello di determinare la strategia ottimale di trasmissione dei nodi mobili sia quando sono cooperativi che quando sono non cooperativi. Quando i nodi sono cooperativi, essi agiscono, come nel caso precedente, in modo da massimizzare la probabilità di consegnare il pacchetto entro il termine temporale sotto vincoli energetici. Nel caso in cui i nodi sono non cooperativi, essi agiscono in modo da massimizzare il loro guadagno atteso nel consegnare il pacchetto sempre considerando i vincoli energetici.
Questo problema è abitualmente modellato come un problema cooperativo / non cooperativo di teoria dei giochi.

Nel seguito, si propone una classificazione dell'attuale stato dell'arte. Identifichiamo tre dimensioni principali:

- *source/mobile*: si classificano i risultati in termini di strategie per la sorgente o per i nodi mobili;
- *singolo pacchetto/multi pacchetto*: classifichiamo i risultati in termini di strategie applicabili quando vi è un solo pacchetto o più pacchetti (nel caso multi pacchetto, distinguiamo tra buffer illimitato e buffer limitato);
- *due hop/multi hop*: classifichiamo i risultati in termini di strategie applicabili solo per le situazioni con due hop (dalla sorgente a un nodo mobile e da questo si trasmette solamente al destinatario) e strategie che riguardano schemi di routing multi hop.

| | | TWO HOP | MULTI HOP |
|---|---|---|---|
| **SINGLE PACK** | | [15],[1],[6] | [14],[33] |
| **MULTI PACK** | limited buf | [20] | |
| | unlimited buf | [24],[26] | |

Tabella 2.1: Classificazione dello stato dell'arte

## 2.2 Lavori principali

Riportiamo nella Tabella 2.1 i principali lavori presenti in letteratura.

In [6], gli autori studiano uno scenario con una sola sorgente e un certo numero di nodi mobili che hanno le stesse capacità di mobilità e nessuna informazione è disponibile ai nodi. Il routing per l'instradamento dei pacchetti è a due hop. Il problema è la determinazione della strategia ottima della sorgente considerando un vincolo energetico sul numero di spedizione dei pacchetti (nessun costo di beaconing viene considerato). Gli autori mostrano che la strategia migliore che la sorgente può attuare è basata su soglia temporale, vale a dire che la sorgente trasmette il pacchetto in tutti gli istanti di tempo che precedono la scadenza temporale e non trasmette nulla dopo tale istante. Viene poi introdotto un algoritmo di stima dei parametri online che permette ai nodi di apprendere la politica ottima nello scenario senza alcuna informazione. Il problema viene poi esteso considerando più classi di nodi in competizione: ogni nodo-giocatore controlla una catena di Markov indipendente di cui conosce solo lo stato. La interazione tra giocatori è dovuta al costo o utilità che dipende dallo stato e dalle azioni di tutti i giocatori. Si dimostra che tale gioco ha un unico equilibrio di Nash che è ottenuto quando ogni classe adotta la propria politica a soglia ottima.

In [15], gli autori si concentrano sui nodi mobili. C'è solo un pacchetto e il routing è a due hop. I nodi sono considerati selfish/non cooperativi

e il problema è di incentivare i nodi mobili a trasmettere il pacchetto. Il primo nodo che consegna il pacchetto riceve un premio unitario. L'unico costo dei nodi mobili è dovuto al beaconing. Il calcolo dell'utilità dipende dalla probabilità (la stessa per ogni nodo) di incontrare il nodo finale e dal costo energetico dovuto al beaconing. Inoltre, tutti i nodi hanno le stesse capacità di mobilità. Gli autori studiano le migliori strategie utilizzando gli strumenti di teoria dei giochi evolutiva.

In [14], gli autori si concentrano su scenari di routing multi hop nel tentativo di progettare incentivi per i nodi mobili in modo da evitare che questi nodi si comportino in modo strategico in termini di edge-hiding e edge-insertion. Il modello proposto funziona solo quando non sono presenti costi di beaconing e ogni nodo mobile è soggetto allo stesso costo di trasmissione. I nodi hanno due tipi di connessione, uno a lungo raggio a bassa frequenza per i messaggi di controllo e uno a corto raggio (intermittente) a banda larga per il trasferimento di dati. Un client che vuole una informazione la può richiedere presso gli AP oppure ad altri nodi che già la possiedono, è inoltre presente una banca per i servizi di pagamento e fornitura delle chiavi. Un simile approccio tecnologico viene proposto in [33], in cui la sfida principale è garantire che la sicurezza dello schema non sia compromessa dagli attacchi di nodi malevoli per ricavare dei vantaggi a discapito delle prestazioni.

In [20], gli autori studiano strategie di routing differenti quando i nodi hanno un buffer limitato e viene fornito un confronto sperimentale delle diverse strategie a seconda della quantità di informazioni della rete che si possiede. La conoscenza di questa informazioni può modificare le prestazioni dell'algoritmo. Viene quindi introdotto un oracolo che è in grado di fornire le informazioni necessarie e a seconda del tipo di oracolo, si identificano tre classi di algoritmi: senza alcuna conoscenza a priori, con conoscenza completa e con conoscenza parziale. Per ogni classe vengono presentati diversi algoritmi di routing che sono poi confrontati per determinare quale sia il migliore compromesso tra quantità di conoscenza

disponibile e prestazioni. Un lavoro simile è presentato in [24], in cui però gli autori introducono alcune informazioni di carattere sociale sui nodi che sono utilizzate per il contatto con altri nodi e per la spedizione del pacchetto. La rete infatti è vista come un grafo di nodi completamente connesso in cui un arco indica il valore tra $[0, 1]$ del legame sociale tra due nodi, questi sono inoltre caratterizzati da un buffer illimitato per i propri pacchetti, ma anche di uno limitato per i pacchetti destinati agli scambi. Anche in questo caso il pacchetto possiede un tempo di vita e una priorità.

Altre opere connesse che si occupano di aspetti cooperativi sono [26] e [27].

Sono stati affrontati anche numerosi problemi correlati a quello principale. In [9] gli autori affrontano il problema di trasferire file di grandi dimensioni dalla sorgente alla destinazione quando non tutti i pacchetti sono disponibili alla sorgente prima dell'inizio della trasmissione.

In [8], viene fornita una classe di meccanismi di replicazione per i pacchetti nella rete che include una codifica per migliorare la probabilità di consegna con successo entro un dato tempo limite.

Ci sono inoltre alcuni lavori che affrontano dei problemi accessori che si possono in qualche modo applicare trasversalmente ad ogni lavoro visto nella classificazione precedente: uno strumento come il Ferry, presentato in [21] permette di garantire la persistenza dei dati con un minore consumo energetico a fronte di un pagamento del servizio; in questo caso bisogna tenere conto della traiettoria del Ferry in modo da ottimizzare le prestazioni totali. In [27] viene anche discussa la possibilità di formare delle coalizioni stabili tra i nodi che abbiano delle caratteristiche in comune con lo scopo di diminuire i tempi e costi di trasmissione del pacchetto. Infatti viene affrontato il problema della cooperazione fra gruppi razionali eterogenei nelle DTN dalla prospettiva dei coalitional game al fine migliorare le prestazioni finali.

## 2.3 Assunzioni Comuni

In questo paragrafo si vuole riassumere quelle che sono le assunzioni comuni alla maggior parte dei lavori:

- i nodi sono caratterizzati da una bassa capacità computazionale ed energetica e per questo motivo sono dotati di un raggio d'azione molto piccolo;
- i costi per i nodi sono legati principalmente al beaconing e al costo per la spedizione dei pacchetti, ma tali costi non vengono considerati per fini pratici;
- la disposizione e la mobilità dei nodi nella rete è modellata come una distribuzione di probabilità sul contatto tra i nodi e può essere di tipo esponenziale o poissoniana;
- i pacchetti sono dotati di un tempo di vita (TTL), oltre il quale l'informazione contenuta nel pacchetto cessa di essere rilevante, e da una priorità nel caso ci sia da effettuare una scelta nella spedizione di un pacchetto nel caso di buffer limitato;
- è uso comune utilizzare dei crediti/rinforzi come premio di consegna per i nodi che consegnano correttamente il pacchetto, ma non risulta chiaro come siano definiti e come sono poi utilizzati a fini pratici;
- nel caso multi hop è spesso coinvolta una entità esterna (banca virtuale) che deve fornire i crediti a tutti i nodi del path oltre che fornire loro le chiavi di cifratura per garantire la corretta identità (problema di autenticazione).

# Capitolo 3

# Descrizione del Problema

In questo capitolo forniamo una descrizione dello scenario in cui ci troviamo, quindi viene formalizzato il modello del problema ed infine vengono enunciate alcune proprietà che saranno utili per la formulazione degli algoritmi.

## 3.1 Scenario

Consideriamo un ambiente popolato da un singolo nodo sorgente, un nodo destinazione e molteplici nodi mobili. Inizialmente un pacchetto è posseduto dalla sorgente e deve essere consegnato al nodo di destinazione entro un tempo $\tau$. Il tempo è discretizzato in quanti di durata fissata $\Delta$ e il numero totale di quanti di tempo disponibili è $K = \lfloor \tau/\Delta \rfloor$, dove il $k$–esimo quanto di tempo corrisponde all'intervallo temporale $[k\Delta, (k+1)\Delta]$. Assumiamo che la trasmissione di un pacchetto occupa un intero quanto di tempo, quindi l'ultimo quanto di tempo utile per ricevere un pacchetto è il $(K-1)$–esimo. I nodi mobili appartengono ad un insieme di classi $C$. Ogni classe racchiude le caratteristiche dei propri nodi, compresi i loro profili di mobilità e le tecnologie di trasmissione. In particolare, data una classe $c \in C$, $N_c$ indica il numero totale di nodi mobili che appartengono a quella classe, $t_c$ indica il tempo di vita locale, vale a dire la quantità

di tempo per il quale ogni nodo della classe $c$ terrà salvata una copia locale del pacchetto prima di scartarlo e quindi non riceverlo nuovamente in futuro. Il profilo di mobilità di un nodo di una classe $c$ è dato dalla velocità media $v_c$. Infine descriviamo la tecnologia di trasmissione per la classe $c$ come:

- $R_c$ indica il range di comunicazione per la classe $c$;
- $\beta_c$ indica il costo di beaconing per la classe $c$ e rappresenta l'energia consumata per il controllo di connessione per sostenere la trasmissione dl pacchetto;
- $\rho_c$ indica il costo di trasmissione per la classe $c$ e rappresenta l'energia consumata per la trasmissione con successo di un pacchetto a un singolo ricevente.

Classi diverse possono avere la stessa tecnologia di trasmissione, per esempio WiFi, e pertanto condividono gli stessi costi di beaconing. Indichiamo con $\omega \in \Omega$ una tecnologia di trasmissione, e indichiamo con $C_\omega \subseteq C$ il sottoinsieme delle classi che utilizzano la tecnologia $\omega$. Indichiamo poi indifferentemente il costo di beaconing $\beta_\omega$ o $\beta_c$ quando $c \in C_\omega$.

Le opportunità di trasmissione tra due nodi nascono quando avviene un contatto che ha luogo quando ogni nodo entra nel range di comunicazione dell'altro. Come già anticipato, ci limiteremo al caso in cui ogni nodo riceve il pacchetto solo dalla sorgente per poi rispedirlo alla corretta destinazione. Assumiamo quindi che i contatti con il nodo sorgente e il nodo destinazione seguono la legge di Poisson multi classe, in cui la frequenza di arrivi per i nodi della classe $c$ (sia per la sorgente che per la destinazione) è indicata da $\lambda_c$ ed è calcolata come indicato in [7]:

$$\lambda_c = \frac{8wR_cv_c}{\pi L^2}$$

dove $w$ è una constante fissata a 1.3693 e $L$ è il raggio del area circolare nella quale si muovono i nodi.

Le opportunità di trasmissione dalla sorgente a un nodo mobile sono gestite dalla politica di spedizione. Quando avviene un contatto tra la sorgente e un nodo mobile che non ha ancora ricevuto il pacchetto, la sorgente invia il pacchetto in accordo con la politica $\boldsymbol{\mu}$ che dipende dal tempo corrente e dalla classe del ricevente. Dati un quanto di tempo $k$ e una classe $c$, il profilo della politica al tempo $k$ è $\boldsymbol{\mu}(k) = (\mu_1(k), \ldots, \mu_{|C|}(k))$ dove $\mu_c(k)$ indica la probabilità di spedizione del pacchetto nel quanto di tempo $k$ per la classe $c$; indichiamo inoltre con $\mu_c$ l'intera politica per la classe $c$. In generale, quando un pacchetto viene spedito, viene consumata una certa quantità di energia e la probabilità di consegna del pacchetto aumenta. Indichiamo con $F_D(\boldsymbol{\mu}, K)$ la probabilità di consegna del messaggio entro l'istante $K\Delta$ dato il profilo delle politiche $\boldsymbol{\mu}$. Ovviamente, tale valore non può crescere indefinitamente a causa del vincolo di budget. Chiamiamo $\Psi$ il limite massimo di energia disponibile per i costi di trasmissione e di beaconing.

## 3.2 Modello Formale

Definiamo $X_{c,k}(\boldsymbol{\mu})$ come la variabile casuale che esprime il numero di nodi mobili di classe $c$ che hanno ricevuto il pacchetto entro il quanto di tempo $k$, mentre $Y_{c,k}(\boldsymbol{\mu})$ è la variabile casuale che esprime il numero di nodi mobili di classe $c$ che hanno ancora una copia del pacchetto al quanto di tempo $k$. Entrambe queste variabili dipendono da $\boldsymbol{\mu}$ e, in generale, sono differenti. Infatti, dato che un nodo mobile può ricevere e poi scartare un pacchetto prima del quanto di tempo $k$, abbiamo che $Y_{c,k}(\boldsymbol{\mu}) \leq X_{c,k}(\boldsymbol{\mu})$. Indichiamo poi con $Q_{c,k,k'}(\mu_c)$ la probabilità che un nodo mobile di classe $c$ non riceva alcun pacchetto nei quanti di tempo $k, \ldots, k'$ in funzione di $\mu_c$. Il numero atteso di nodi mobili di classe $c$ che hanno ricevuto un pacchetto nei quanti di tempo $0, \ldots, k$ è:

$$\mathbb{E}[X_{c,k}(\boldsymbol{\mu})] = N_c \cdot (1 - Q_{c,0,k}(\mu_c))$$

dove

$$Q_{c,k,k'}(\mu_c) = e^{-\lambda_c \Delta \sum_{i=k}^{k'} \mu_c(i)}$$

Il numero atteso di nodi mobili di classe $c$ che hanno ancora una copia del pacchetto nel quanto di tempo $k$ è:

$$\mathbb{E}[Y_{c,k}(\mu_c)] = N_c \cdot (1 - Q_{c,\max\{0,k-t_c\},k}(\mu_c))$$

La probabilità che un pacchetto venga consegnato alla corretta destinazione entro l'istante $k\Delta$ è:

$$F_D(\boldsymbol{\mu}, k) = 1 - \prod_{c \in C} \prod_{h=0}^{k-1} X^*_{c,h}(\lambda_c \Delta, \boldsymbol{\mu})$$

dove

$$X^*_{c,h}(s, \boldsymbol{\mu}) = \mathbb{E}[e^{-sY_{c,h}(\boldsymbol{\mu})}]$$

è la trasformata di Laplace-Stieltjes di $X_{c,k}$.
Il vincolo di budget energetico è formulato come:

$$\sum_{c \in C} \rho_c N_c (1 - Q_{c,0,K}(\mu_c)) + \sum_{\omega \in \Omega} \sum_{k=0}^{K-1} \beta_\omega \cdot \left(1 - \prod_{c \in C_\omega} \Big(1 - \mu_c(k)\Big)\right) \leq \Psi \quad (3.1)$$

Il termine a sinistra della diseguaglianza prende in considerazione i costi attesi di trasmissione del pacchetto e i costi attesi di beaconing per la classe $c$, dato un profilo di politiche $\boldsymbol{\mu}$. In particolare, i costi di trasmissione sono ottenuti moltiplicano per $\rho_c$ il numero atteso di nodi mobili che riceveranno il pacchetto nei quanti di tempo tra $0, \ldots, K$; mentre viene pagato un costo di beaconing pari a $\beta_\omega$ per ogni quanto di tempo $h$ con probabilità $\mu_c(h)$, ad esempio, quando può avvenire la trasmissione di un pacchetto, e per ogni tecnologia $\omega$.
Il problema che ci proponiamo di risolvere è la determinazione della politica ottima $\boldsymbol{\mu}^*$, vale ad dire quella politica che massimizza la probabilità di consegna del pacchetto $F_D(\boldsymbol{\mu}, K)$ senza violare il vincolo di budget (3.1).

| | |
|---|---|
| $\tau$ | tempo di vita del pacchetto |
| $\Delta$ | durata di ogni quanto di tempo |
| $K$ | numero di quanti disponibili |
| $k$ | generico quanto di tempo |
| $C$ | insieme delle classi di nodi mobili |
| $c$ | generica classe di nodi mobili |
| $N_c$ | numero di nodi mobili della classe $c$ |
| $R_c$ | range di trasmissione della classe $c$ |
| $v_c$ | velocità della classe $c$ |
| $L$ | raggio dell'area circolare in cui si muovono i nodi |
| $w$ | costante |
| $\lambda_c$ | frequenza di arrivi per la classe $c$ |
| $\mu_c(k)$ | politica di spedizione per la classe $c$ al quanto di tempo $k$ |
| $\mu_c$ | politica di spedizione per la classe $c$ |
| $\boldsymbol{\mu}(k)$ | profilo della politica di spedizione al quanto di tempo $k$ |
| $\boldsymbol{\mu}$ | profilo della politica di spedizione |
| $h_c$ | valore di soglia per la politica della classe $c$ |
| $F_D$ | c.d.f. del ritardo della probabilità di consegna |
| $\Psi$ | budget energetico a disposizione |
| $\epsilon$ | fattore di discretizzazione per ogni quanto di tempo |
| $X_{c,k}(\boldsymbol{\mu})$ | nodi infetti della classe $c$ al quanto $k$ considerando $\boldsymbol{\mu}$ |
| $X^*_{c,k}(s, \boldsymbol{\mu})$ | trasformata di Laplace-Stieltjes di $X_{c,k}(\boldsymbol{\mu})$ |
| $Q_{c,k,k'}(\mu_c)$ | prob. che un nodo di $c$ non venga infettato in $k, \ldots, k'$ |

Tabella 3.1: Notazione utilizzata

## 3.3 Proprietà del Problema

Mostriamo ora alcune proprietà teoriche che saranno utili nell'affrontare il problema di ottimizzazione.

**Proprietà 3.3.1.** *Le politiche ottime o consumano completamente il budget o prescrivono che tutte le classi trasmettano per tutti i quanti di tempo.*

*Dimostrazione.* E' facile vedere che la funzione obiettivo $F_D(\boldsymbol{\mu}, K)$ è monotona crescente in $\sum_{h=0}^{K-1} \mu_c(h)$ e che, di conseguenza, trasmettere per un numero maggiore (in atteso) di quanti di tempo non può portare ad una probabilità di consegna del pacchetto inferiore. □

In modo simile a quanto proposto in [7], definiamo la politica a soglia $\mu_c$ come:

$$\mu_c(k) = \begin{cases} 1 & k < h_c \\ \alpha & k = h_c \\ 0 & k > h_c \end{cases}$$

dove $\alpha \in [0, 1)$. Come nel caso di una singola classe, le politiche ottime sono quelle a soglia.

**Proprietà 3.3.2.** *Le politiche ottime sono quella a soglia.*

*Dimostrazione.* La c.d.f. del ritardo di consegna è $F_D(\boldsymbol{\mu}, K) = 1 - \prod_c \Gamma_c(s)$, dove $\Gamma_c(s) = \prod_{h=0}^{K-1} X^*_{c,h}(s)$ e $s = \lambda_c \Delta$. Indichiamo con $\mu_c$ una politica non a soglia per la classe $c$ e con $\hat{\mu}_c$ la politica ottenuta spostando a sinistra tutti i quanti di tempo non vuoti di $\mu_c$ e arrotondandoli in modo che $\hat{\mu}_c$ segua la definizione di politica a soglia introdotta prima. Per ogni $(\mu_c, \hat{\mu}_c)$ ottenuta in questo modo, abbiamo che $\Gamma_c(s) \geq \hat{\Gamma}_c(s)$ e quindi

$$1 - \Gamma_c(s) \cdot \Gamma_{-c}(s) \leq 1 - \hat{\Gamma}_c(s) \cdot \Gamma_{-c}(s)$$

cioè, per ogni politica congiunta $\boldsymbol{\mu}$, se sostituiamo la politica marginale di una classe $c$ con la sua versione a soglia, la probabilità di consegna entro $K$ quanti di tempo non decresce. □

**Proprietà 3.3.3.** *Le politiche ottime possono assegnare soglie non intere ad ogni classe.*

*Dimostrazione.* Consideriamo, per esempio, una rete con due classi con: $K = 20$, $\Delta = 100$, $\Psi = 0.7$, $N_1 = 1$, $N_2 = 2$, $\lambda_1 = 21 \times 10^{-5}$, $\lambda_2 = 20 \times 10^{-5}, t_1 = t_2 = K$. Approssimiamo il profilo della politica ottima discretizzando il valore di $h_c$ con una griglia fine di passo 0.01. In aggiunta a ciò, abbiamo considerato tutti i punti in cui la soglia di una classe è intera mentre la soglia dell'altra classe è calcolata in modo che tutto il vincolo di budget sia consumato. Valutiamo la funzione obiettivo in tutti questi punti e selezioniamo il massimo. La politica ottima approssimata è $h_1 = 7.87$, $h_2 = 15.99$. La soluzione ottima assegna quindi una parte frazionaria a entrambe le classi. □

**Proprietà 3.3.4.** *Il problema di ottimizzazione è non-lineare e non-convesso.*

*Dimostrazione.* La non-linearità è banale. La non-convessità si dimostra mostrando la non-convessità della regione di ammissibilità calcolando la matrice Hessiana del vincolo di budget (3.1) alla quale ci riferiamo qui con $u$ (da notare che da ora utilizzeremo sempre politiche a soglia). La matrice Hessiana $\mathcal{H}$ è:

$$\mathcal{H} = \begin{bmatrix} \frac{\partial^2 u}{\partial h_1^2} & & 0 \\ & \ddots & \\ 0 & & \frac{\partial^2 u}{\partial h_{|C|}^2} \end{bmatrix} = \begin{bmatrix} -N_1\lambda_1^2\Delta^2 e^{-\lambda_1 \Delta h_1} & & 0 \\ & \ddots & \\ 0 & & -N_{|C|}\lambda_{|C|}^2\Delta^2 e^{-\lambda_{|C|}\Delta h_{|C|}} \end{bmatrix}$$

Si può facilmente vedere che tutti gli autovalori sono strettamente negativi per ogni profilo di politiche $\boldsymbol{\mu}$ e quindi la regione di ammissibilità è non convessa. □

Queste proprietà mostrano che il problema di ottimizzazione è molto complicato da risolvere. In particolare, l'adozione di tecniche di programmazione non-convesse richieste dalla natura del problema non garantisce che venga trovata la soluzione ottima globale. Per queste ragioni, ci concentriamo sul problema di sviluppare algoritmi di approssimazione e studiare i loro errori di approssimazione teorici ed empirici.

# Capitolo 4

# Algoritmi di approssimazione

In questo capitolo introduciamo due algoritmi di approssimazione per calcolare la politica (a soglia) ottima e discutiamo di come essi garantiscono sulla perdita di qualità della soluzione.

## 4.1 Algoritmo di approssimazione a tempo non polinomiale

Partiamo col definire uno schema di approssimazione che non lavora in tempo polinomiale, ma per il quale la perdita di ottimalità può essere limitata arbitrariamente ottenendo una soluzione che è ottima eccetto per valori numerici trascurabili. Aggiungiamo un ulteriore vincolo al problema di ottimizzazione, facendo in modo che solo la politica di una singola classe possa avere soglia frazionaria, mentre tutte le altre hanno soglie intere. Questo vincolo riduce la qualità della soluzione (vedi Proprietà 3.3.3), ma consente di avere una versione combinatoria del problema di ottimizzazione. Infatti, una volta che a tutte le classi tranne una sono state assegnate politiche intere, la politica potenzialmente frazionaria della classe restante è determinata in modo univoco o dalla politica che consuma tutto il bilancio rimanente o da quella che trasmette fino all'ultimo quanto di tempo (vedi Proprietà 3.3.1). Inoltre, dividiamo ogni quanto di tempo di lunghez-

za $\Delta$ in sub-slot di lunghezza $\epsilon\Delta$ dove, per semplicità, $\frac{1}{\epsilon} \in \mathbb{N}$. Possiamo dare una interpretazione probabilistica delle politiche che prescrivono di trasmettere solo per alcuni sub-slot di un dato quanto di tempo. Cioè, se la trasmissione avviene, per esempio, solo su $z$ sub-slot su $\epsilon\Delta$, in un quanto si tempo, allora in tale quanto avremo una probabilità di trasmissione pari a $z\epsilon$.

Questo nuovo problema può essere risolto ottimamente utilizzando un semplice algoritmo di enumerazione: numeriamo tutte le possibili politiche a soglia del problema e selezioniamo la migliore (vedi Proprietà 3.3.2). L'Algoritmo 1 riporta i passi necessari. Al Passo 1, l'algoritmo inizializza $F^*$ a zero. Se non è possibile consumare completamente il budget disponibile, allora la politica ottimale è assegnare $h_c = (K/\epsilon) - 1$ ad ogni classe $c$ (Passo 2–3). In caso contrario, l'algoritmo ordina tutte le classi $c$ in modo lessicografico e, per ogni classe $c$, enumera tutta i profili delle politiche $\boldsymbol{\mu} = (\mu_c, \boldsymbol{\mu}_{-c})$ t.c. $\boldsymbol{\mu}_{-c}$ è intera e il budget $\Psi$ è consumato interamente (Passo 5–6). Infine, teniamo traccia della migliore politica trovata fino a quel momento.

**Algorithm 1** $\epsilon$ - grid search

1: $F^* \leftarrow 0$
2: **if** $\boldsymbol{\mu}$ s.t. $h_c = \frac{K}{\epsilon} - 1$ for all $c$ is feasible **then**
3: $\quad \boldsymbol{\mu}^* = \boldsymbol{\mu}$
4: **else**
5: $\quad$ **for** $c \in C$ **do**
6: $\quad\quad$ **for** every $\boldsymbol{\mu} = (\mu_c, \boldsymbol{\mu}_{-c})$ s.t. $\boldsymbol{\mu}_{-c}$ is integer and budget $\Psi$ is entirely consumed **do**
7: $\quad\quad\quad$ **if** $F_D(\frac{K}{\epsilon} - 1, \boldsymbol{\mu}) > F^*$ **then**
8: $\quad\quad\quad\quad \boldsymbol{\mu}^* \leftarrow \boldsymbol{\mu}$
9: $\quad\quad\quad\quad F^* \leftarrow F_D(\frac{K}{\epsilon} - 1, \boldsymbol{\mu})$
10: $\quad\quad\quad$ **end if**
11: $\quad\quad$ **end for**
12: $\quad$ **end for**
13: **end if**

Descriviamo ora un efficiente metodo per enumerare tutti e solo i profili delle politiche ammissibili $\boldsymbol{\mu} = (\mu_c, \boldsymbol{\mu}_{-c})$ t.c. $\boldsymbol{\mu}_{-c}$ è intero e il budget $\Psi$ è consumato per intero. Per prima cosa, costruiamo un ordine lessicografico su $C_{-c} = C \setminus c$ e selezioniamo in ordine lessicografico le classi in $C_{-c}$. Poi, per ogni $c' \in C_{-c}$ determiniamo il range dei valori possibili per $h_{c'}$ sulla base delle politiche già assegnate alle classi $c'' \succ c'$ nel modo seguente:

$$I_{c'}(\boldsymbol{\mu}_{-c'}) = \left\{ \max\left\{0, \lceil r_{c'}(\overline{\boldsymbol{\mu}}_{-c'}) \rceil\right\}, \ldots, \min\left\{ \frac{K}{\epsilon} - 1, \lfloor r_{c'}(\underline{\boldsymbol{\mu}}_{-c'}) \rfloor \right\} \right\} \tag{4.1}$$

dove $r_{c'}(\boldsymbol{\mu}_{-c'})$ è calcolato come segue:

- inizialmente calcoliamo:

$$r_{c'}(\boldsymbol{\mu}_{-c'}) = -\frac{\log\left( \frac{N_{c'} + A - \frac{\Psi}{\rho} + \max\limits_{c'':c'' \neq c'} \{\frac{\beta}{\rho} h_{c''}\}}{N_{c'}} \right)}{\lambda_{c'} \Delta\epsilon} \tag{4.2}$$

  dove

$$A = \sum_{c'':c'' \neq c'} N_{c''} \left( 1 - e^{-\lambda_{c''} \Delta\epsilon \sum\limits_{k=0}^{\frac{K}{\epsilon}-1} \mu_{c''}(k)} \right) \tag{4.3}$$

- se $r_{c'}(\boldsymbol{\mu}_{-c'}) > \max\limits_{c'':c'' \neq c'} \{\frac{\beta\epsilon}{\rho} h_{c''}\}$, allora $r_{c'}(\boldsymbol{\mu}_{-c'})$ è la soluzione (che può essere approssimata utilizzando il metodo di Newton) della seguente equazione:

$$N_{c'}(1 - e^{-\lambda_c \epsilon \Delta r_{c'}(\boldsymbol{\mu}_{-c'})}) + \sum_{c'':c'' \neq c'} N_{c''} \left( 1 - e^{-\lambda_{c''} \Delta\epsilon \sum\limits_{k=0}^{\frac{K}{\epsilon}-1} \mu_{c''}(k)} \right) - \frac{\Psi}{\rho} + \frac{\beta\epsilon}{\rho} r_{c'}(\boldsymbol{\mu}_{-c'}) = 0$$

e dove $\overline{\boldsymbol{\mu}}_{-c'}$ e $\underline{\boldsymbol{\mu}}_{-c'}$ sono definiti nel seguente modo:

$$\overline{\boldsymbol{\mu}}_{-c'} = \begin{cases} \overline{\mu}_{c''} = \mu_{c''} & c'' \succ c' \\ \overline{\mu}_{c''} : h_{c''} = \frac{K}{\epsilon} - 1 & c' \succ c'' \\ \overline{\mu}_{c} : h_{c} = \frac{K}{\epsilon} - 1 \end{cases}$$

$$\underline{\boldsymbol{\mu}}_{-c'} = \begin{cases} \underline{\mu}_{c''} = \mu_{c''} & c'' \succ c' \\ \underline{\mu}_{c''} : h_{c''} = 0 & c' \succ c'' \\ \underline{\mu}_{c} : h_{c} = 0 \end{cases}$$

Una volta eseguiti i passi precedenti, per ogni elemento in $I_{c'}(\boldsymbol{\mu}_{-c'})$, assegniamo esso a $h_{c'}$ e passiamo alla classe successiva secondo l'ordine lessicografico stabilito. Infine, una volta che tutte le politiche delle classi $c' \in C_{-c}$ sono state assegnate, la politica di $c$ si ottiene facilmente da $h_c = r_c(\boldsymbol{\mu}_{-c})$.

**Teorema 4.1.1.** *Il precedente metodo enumera tutte e sole le politiche ammissibili che consumano esattamente l'intero budget, nelle quali al massimo un $h_c$ è frazionario.*

*Dimostrazione.* Dobbiamo provare che:

- tutte le politiche eccetto $\mu_c$ sono intere,
- il budget è consumato interamente,
- sono enumerate tutte e sole le politiche ammissibili.

I primi due punti sono banali per costruzione (dato che la politica della classe $c$ è la sola potenzialmente non intera ed è calcolata come la politica che consuma il budget residuo date le politiche (intere) di tutte le altre classi). Per dimostrare il terzo punto, osserviamo che l'intervallo $I$ è sempre ben definito. Infatti $\lfloor r_{c'}(\underline{\boldsymbol{\mu}}_{-c'}) \rfloor$ restituisce il maggior $h_{c'}$ che consuma

esattamente il budget residuo dato il consumo di budget di tutte le altre classi che hanno precedenza su $c'$ in ordine lessicografico e assumendo che le classi che seguono non trasmettono mai. Assegnando una politica maggiore di $\min\left\{\frac{K}{\epsilon}-1, \lfloor r_{c'}(\underline{\boldsymbol{\mu}}_{-c'})\rfloor\right\}$ non è possibile consumare esattamente l'intero budget (viene consumato più del dovuto). Se le politiche assegnate alle classi con precedenza sono ammissibili, allora $\lfloor r_{c'}(\underline{\boldsymbol{\mu}}_{-c'})\rfloor$ è sempre non negativo. Allo stesso modo, $\lceil r_{c'}(\overline{\boldsymbol{\mu}}_{-c'}\rceil$ restituisce il minor $h_{c'}$ che consuma esattamente il budget residuo dato il consumo di budget di tutte le altre classi che hanno precedenza su $c'$ in ordine lessicografico e assumendo che le classi che seguono trasmettono fino all'ultimo quanto di tempo disponibile (cioè fino a $\frac{K}{\epsilon}-1$). Assegnando una politica minore di $\max\left\{0, \lceil r_{c'}(\overline{\boldsymbol{\mu}}_{-c'})\rceil\right\}$ non è possibile consumare l'intero budget (la soluzione non è ottima). Anche in questo caso, se le politiche assegnate alle classi che precedono $c'$ sono ammissibili, allora $\lceil r_{c'}(\overline{\boldsymbol{\mu}}_{-c'}\rceil$ è sempre minore di $\frac{K}{\epsilon}-1$. Quindi, per costruzione, per ogni politica appartenente all'intervallo $I$ assegnata alla classe $c'$, è sempre possibile trovare una politica ammissibile per la classe successiva. □

Il numero di politiche enumerate dal Algoritmo 1 è esponenziale nel numero di classi, cioè $O((\frac{K}{\epsilon}-1)^{|C|-1})$.

Possiamo derivare un lower bound teorico sulla qualità delle soluzioni trovate dall'Algoritmo 1 rispetto alla soluzione ottima del problema di ottimizzazione.

**Teorema 4.1.2.** *Chiamo $\tilde{F}_D$ il valore della soluzione trovata dall'Algoritmo 1 e $F_D^*$ il valore della soluzione ottima, ho che* $\dfrac{\tilde{F}_D}{F_D^*} \geq \dfrac{1-(\frac{1}{2})^{\frac{K}{\epsilon}-1}}{1-(\frac{1}{2})^{|C|\frac{K}{\epsilon}-1}}$.

*Dimostrazione.* Chiamo $\boldsymbol{\mu}^*$ il profilo della politica ottima e chiamo $\tilde{\boldsymbol{\mu}}_c$ il profilo della politica in cui $\tilde{h}_{c'} = \lfloor h_{c'}^*\rfloor$ per ogni $c' \neq c$ e $\tilde{h}_c = h_c^*$. Ovviamente, $F_D^* \geq F_D(K/\epsilon, \tilde{\boldsymbol{\mu}}_c)$. Inoltre, è ovvio che $F_D^* \geq \tilde{F}_D \geq \max_c\{F_D(K, \tilde{\boldsymbol{\mu}}_c)\}$. Questo perché $\tilde{\boldsymbol{\mu}}_c$ è un profilo di politica ammissibile, in cui al massimo una politica è frazionaria, che non garantisce di consumare interamente il

budget a disposizione. Possiamo ricavare un lower bound per $F_D\left(\frac{K}{\epsilon}, \tilde{\boldsymbol{\mu}}_c\right)$ come:

$$F_D\left(\frac{K}{\epsilon}, \tilde{\boldsymbol{\mu}}_c\right) = 1 - \prod_{c' \in C} \prod_{k=0}^{\frac{K}{\epsilon}-1} X^*_{c',k}(\lambda_{c'}\epsilon\Delta, \tilde{\boldsymbol{\mu}}_c) \geq 1 - \prod_{k=0}^{\frac{K}{\epsilon}-1} X^*_{c,k}(\lambda_c\epsilon\Delta, \tilde{\boldsymbol{\mu}}_c) \tag{4.4}$$

Usando questo lower bound su $F_D\left(\frac{K}{\epsilon}, \tilde{\boldsymbol{\mu}}_c\right)$, possiamo scrivere:

$$\frac{\tilde{F}_D}{F^*_D} \geq \max_c \left\{ \frac{1 - \prod_{k=0}^{\frac{K}{\epsilon}-1} X^*_{c,k}(\lambda_c\epsilon\Delta, \boldsymbol{\mu}^*)}{1 - \prod_{c' \in C} \prod_{k=0}^{\frac{K}{\epsilon}-1} X^*_{c',k}(\lambda_{c'}\epsilon\Delta, \boldsymbol{\mu}^*)} \right\}$$

dato che abbiamo $\tilde{h}_c = h^*_c$, avendo $\tilde{\boldsymbol{\mu}}_c$ e $\boldsymbol{\mu}^*$. Quindi siamo interessati a:

$$\min \max_c \left\{ \frac{1 - \prod_{k=0}^{\frac{K-1}{\epsilon}} X^*_{c,k}(\lambda_c\epsilon\Delta, \boldsymbol{\mu}^*)}{1 - \prod_{c' \in C} \prod_{k=0}^{\frac{K-1}{\epsilon}} X^*_{c',k}(\lambda_{c'}\epsilon\Delta, \boldsymbol{\mu}^*)} \right\}$$

dove la minimizzazione è su tutti i parametri. Nonostante la definizione di $X^*$ sia intricata, un bound può essere trovato trascurando la natura esponenziale di tutti gli $X^*$ e considerandoli come valori arbitrari in $[0, 1]$. In questo caso, per ragioni di simmetria, i valori che minimizzano il rapporto massimo devono essere $X^*_{c,k} = \frac{1}{2}$ per ogni classe $c$, ottenendo la tesi del teorema. □

Si noti che il lower bound teorico non dipende dalla presenza o meno dei costi di beaconing. Il caso pessimo è quando $K = 1$ e $|C| \to \infty$, ottenendo un rapporto di $1 - \frac{1}{2}^{\frac{1}{\epsilon}}$. Ad ogni modo, si può osservare che il rapporto nel caso pessimo tende a uno in modo esponenziale in $\frac{1}{\epsilon}$. Quindi possiamo ottenere un buon rapporto di approssimazione con valori piccoli di $\frac{1}{\epsilon}$, per esempio, il lower bound teorico sul rapporto di approssimazione è circa $1 - 10^{-4}$ quando $\frac{1}{\epsilon} = 10$. L'Algoritmo 1 è uno schema di approssimazione (AS), dato che il rapporto tende a 1 come $\epsilon$ tende a 0. Comunque non è un AS a tempo completamente polinomiale (FPTAS), dato che la sua complessità non è polinomiale in tutti i parametri.

Riportiamo in Appendice A.1 il codice MATLAB che abbiamo utilizzato per questo algoritmo.

## 4.2 Algoritmo di approssimazione a tempo polinomiale

In questa sezione, discutiamo di un approccio euristico per approssimare la politica ottima in tempo polinomiale. Iniziamo fornendo l'Algoritmo 2, un metodo che massimizza in modo greedy una funzione obiettivo $G_i$. Considereremo due versioni di questa funzione, indicate con $G_1$ e $G_2$, e discuteremo i bound di approssimazione garantiti dal loro impiego.

**Algorithm 2** $i$ - greedy construction

$\mu_1(k), \ldots, \mu_C(k) \leftarrow 0 \; \forall k$
$k_1, \ldots, k_C \leftarrow 0$
$F^* \leftarrow 0$
**while** Constraint (3.1) is satisfied **do**
    **for** every class $c$ **do**
        $\mu_c(h_c + 1) \leftarrow \min\{1, r_c(\boldsymbol{\mu})\}$
        $\delta_c \leftarrow G_i(\boldsymbol{\mu})$
        $\mu_c(k_c + 1) \leftarrow 0$
    **end for**
    $c^* \leftarrow \arg\max_{c \in C}\{\delta_c\}$
    $\mu_{c^*}(k_{c^*} + 1) \leftarrow \min\{1, \hat{\mu}_{c^*}\}$
    $k_{c^*} \leftarrow k_{c^*} + 1$
    $F^* \leftarrow F^* + \delta_{c^*}$
**end while**

L'Algoritmo 2 funziona con la stessa rappresentazione discreta del tempo che abbiamo introdotto in precedenza, in cui ogni quanto di tempo ha una durata temporale di $\epsilon\Delta$. Si parte da una politica iniziale vuota e si

procede considerando solo politiche a soglia intere. Ad ogni iterazione, si aggiunge un quanto di tempo localmente ottimo per una classe $c$, il che significa che tale classe trasmetterà con probabilità pari a 1 per un ulteriore quanto di tempo aggiuntivo. La classe $c$ selezionata è quella che introdurrebbe il guadagno più grande in $G_i$ se il quanto di tempo le venisse assegnato. Indichiamo con $k_c$ l'indice intero per la classe $c$, riferito all'ultimo quanto di tempo assegnato. Allo stesso modo, $\delta_c$ indica il guadagno marginale discreto di $G_i$ ottenuto assegnando un quanto di tempo alla classe $c$ nella politica corrente.

### 4.2.1 Prima versione, ottimizzazione locale di $F_D$

Nella prima versione dell'Algoritmo 2, consideriamo la massimizzazione del guadagno marginale di $F_D$, ovvero la probabilità di consegna. Vale a dire, al passo (7) abbiamo che $G_1(\boldsymbol{\mu}) = F_D(K, \boldsymbol{\mu}) - F^*$. In questo caso, $\delta_c$ rappresenta il beneficio, in termini di probabilità di consegna, che un quanto di tempo aggiuntivo per la classe $c$ introdurrebbe nell'iterazione corrente. Sfruttando un risultato presentato in [22] siamo in grado di fornire un bound sulla qualità della soluzione ottenuta con questa versione dell'algoritmo greedy. Il risultato che useremo può essere riassunto come segue (vedi [22] per i dettagli).

**Teorema 4.2.1** (Da [22])**.** *Dato un insieme base $\Xi$, una set function $F : 2^{\Xi} \to \mathbb{R}$, e un intero positivo $W \in \mathbb{N}^+$, consideriamo il problema di trovare $\hat{S}^* = \arg\max_{S \subseteq \Xi, |S| \leq W} F(S)$. Allora se $F$ è sub-modulare, abbiamo che per ogni intero $0 \leq l \leq W$*

$$F(S_l) \geq (1 - e^{-l/W}) F(\hat{S}^*)$$

*dove $S_l \in \Xi$ è l'insieme costruito dopo $l$ iterazioni della seguente regola greedy di selezione elemento:*

$$S_i = \begin{cases} \emptyset & \textit{if } i{=}0 \\ S_{i-1} \cup \arg\max_{s \in \Xi} F(S_{i-1} \cup \{s\}) & \textit{else} \end{cases} \tag{4.5}$$

Il Teorema 4.2.1 afferma che massimizzando in modo greedy una set function sub-modulare, si arriva ad una certa sub-ottimalità. Alla fine, il bound converge a $(1 - \frac{1}{e})$ ($\approx 0.63$) quando $l = W$, cioè quando viene fatto il numero massimo di selezioni permesse dal vincolo di cardinalità.

Per applicare questo risultato all'Algoritmo 2, dobbiamo dimostrare che il problema di trovare una politica intera ottima può essere espresso come la massimizzazione di una set function sub-modulare soggetta a un vincolo di cardinalità. Questa similarità può essere mostrata utilizzando il seguente semplice formalismo. Supponiamo che ciascun elemento $e \in \Xi$ nell'insieme base è una coppia $(c, k)$ dove $c \in C$ e $k \in \{1 \dots \frac{K}{\epsilon} - 1\}$. Poi, ogni sottoinsieme $S \subseteq \Xi$ può essere associato in modo univoco ad una politica intera che indichiamo con $\mu^S$. Intuitivamente, la corrispondenza tra $S$ e $\mu^S$ si ottiene dalla seguente regola di costruzione:

$$\mu_c^S(k) = \begin{cases} 1 & (c, k) \in S \\ 0 & \text{else} \end{cases}$$

Quindi, la funzione obiettivo per una politica $\mu^S$ può essere riscritta come una set function $F(K, S)$.

Il secondo passo necessario è quello di ricavare un vincolo di cardinalità per definire la regione di ammissibilità del problema. Nel nostro problema, l'ammissibilità di una politica è determinata dal limite di budget, cioè dal vincolo (3.1). Per questo motivo, idealmente si vorrebbe trovare un $W$ tale che $|S| > W$ se e solo se $\mu^S$ viola il Vincolo (3.1). Tuttavia, si può facilmente mostrare che la ammissibilità del budget non può essere espressa con un vincolo di cardinalità. Il motivo è semplice. Il budget di una politica non dipende solo dal numero di quanti di tempo in cui si trasmette, ma anche da come questi quanti di tempo sono distribuiti tra

le varie classi di nodi. Ad ogni modo, un necessario (ma non sufficiente) upper bound per la cardinalità può essere determinato tramite il seguente teorema.

**Teorema 4.2.2.** *Ogni politica a soglia intera ammissibile non può assegnare la massima probabilità di trasmissione a più di*

$$W = \min\{\max_c\{r_c(\boldsymbol{\mu}^{\emptyset})\}, \frac{K}{\epsilon} - 1\}, \tag{4.6}$$

*dove* $\boldsymbol{\mu}^{\emptyset}$ *è la politica vuota.*

*Dimostrazione.* Assumiamo che $\hat{c} = \arg\max_c\{r_c(\boldsymbol{\mu}^{\emptyset})\}$. Poi, consideriamo una politica a soglia $\mu^S$ dove $|S| > W$. Se $\mu^S$ non supera il budget allora, per definizione, la politica così ottenuta dovrebbe essere ammissibile a sua volta: per ogni $(c, k) \in S$ dove $c \neq \hat{c}$ sostituisce $(c, k)$ con $(\hat{c}, h_{\hat{c}} + 1)$. Tuttavia, per definizione di $W$ tale politica non può rispettare il vincolo di budget. □

Con le ipotesi fatte sopra, il problema della politica intera ottima può essere associato, mediante un rilassamento del vincolo di ammissibilità, alla massimizzazione della set function $F_D(K, S)$, soggetta a $|S| \leq W$. Nel prossimo passaggio mostriamo la sub-modularità di $F_D$.

**Proprietà 4.2.3.** *La set function* $F$ *è sub-modulare rispetto a* $\Xi$.

*Dimostrazione.* Per prima cosa, Consideriamo una istanza con una sola classe. Grazie alla Proprietà 3.3.2, possiamo concentrarci solo sulle politiche a soglia e riscrivere $F$ come funzione di $h$, cioè il valore di soglia (che in generale può essere non intero). Quindi si può facilmente mostrare che $F(h)$ è una funzione concava dato che la matrice Hessiana ha autovalori strettamente negativi. Data la funzione $f : \mathbb{N} \to \mathbb{R}^+$, allora $f(|S|)$ è sub-modulare sul sottoinsieme $S$ di un arbitrario insieme $\Xi$ se e solo se $f$ è concava. Possiamo concludere che $F$ è sub-modulare nel caso di singola classe. Mostriamo ora la sub-modularità nel caso di due

classi nella rete. Indichiamo con $\Delta F(S|e)$ il guadagno marginale di $F$ ottenuto con l'aggiunta di un elemento $e$ all'insieme $S$, cioè aggiungiamo un quanto di trasmissione alla politica $\mu^S$ di una qualche classe. Per mantenere la sub-modularità, dobbiamo dimostrare che per ogni $S_a$, $S_b$, $e$ tale che $S_a \subseteq S_b \subset \Xi$ e $e \in \Xi \setminus S_b$ abbiamo che $\Delta F(S_a|e) \geq \Delta F(S_b|e)$. Per definizione $e$ aggiunge un quanto di trasmissione ad una singola classe, assumiamo senza perdita di generalità che questa classe è $c_1$. Allora abbiamo che:

$$\begin{aligned}\Delta F(S_a|e) = [1 - (1 - (F_{c_1}(S) + \Delta F_{c_1}(S_a|e)))(1 - F_{c_2}(S))] \\ - [1 - (1 - F_{c_1}(S))(1 - F_{c_2}(S))] \\ = (1 - F_{c_2}(S_a))\Delta F_{c_1}(S_a|e)\end{aligned}$$

e, in modo analogo,

$$\Delta F(S_b|e) = (1 - F_{c_2}(S_b))\Delta F_{c_1}(S_b|e)$$

Dato che $\Delta F_{c_1}(S_a|e) \geq \Delta F_{c_1}(S_b|e)$ per la sub-modularità di $F_{c_1}$ e $F_{c_2}(S_b) \geq F_{c_2}(S_a)$ per la monotonicità di $F_{c_2}$, abbiamo che $F$ è sub-modulare. Lo stesso ragionamento può essere esteso ad un numero arbitrario di classi. □

Il Teorema 4.2.1 può essere applicato come mostra l'algoritmo 2 che corrisponde alla regola greedy (4.5) di selezione di un elemento. È facile vedere che tale regola (4.5), quando applicata al problema con politiche intere, procede scegliendo l'ottimo locale allo stesso modo che nell'Algoritmo 2. Quindi, siamo ora nella posizione di enunciare il seguente teorema:

**Teorema 4.2.4.** *Indichiamo con $S^*$ la politica fornita dall'Algoritmo 1 e con $S_l^1$ la politica costruita dall'Algoritmo 2 (versione 1) dopo l iterazioni. Abbiamo quindi che $F_D(K, S_l^1) \geq (1 - e^{-l/W})F_D(K, S^*)$.*

*Dimostrazione.* La disuguaglianza indicata nel teorema segue immediatamente dalle due seguenti proprietà. Primo, applicando il Teorema 4.2.1 all'Algoritmo 2 (versione 1) abbiamo che $F_D(K, S_l^1) \geq (1-e^{-l/W})F_D(K, \hat{S}^*)$. Secondo, dato che $\hat{S}^*$ è la soluzione ottima della versione rilassata del problema con politiche intere, vale che $F_D(K, S^*) \leq F_D(K, \hat{S}^*)$. □

Il teorema precedente fornisce una bound online sulla qualità della soluzione, essendo dipendente dal numero di iterazioni che l'algoritmo riuscirà ad effettuare senza violare il vincolo di budget. Un bound garantito offline può essere dato calcolando il numero minimo di quanti di tempo $s_c$ da assegnare a ciascuna classe $c$. Questo numero può essere calcolato impostando $\mu_{c'}(i) = 1 \; \forall i \; 0 \leq i \leq K, c' \neq c$ e calcolando il numero massimo di quanti di tempo durante il quale $c$ può trasmettere senza saturare il budget.

**Corollario 4.2.5.** *Per ogni soluzione $\hat{S}^1$ ottenuta mediante l'Algoritmo 2 (versione 1) abbiamo che $F_D(K, \hat{S}^1) \geq (1 - e^{-\sum_{c \in C} s_c/W})F_D(K, S^*)$.*

Riportiamo in Appendice A.2 il codice MATLAB che abbiamo utilizzato per questo algoritmo.

### 4.2.2 Seconda versione, normalizzazione di $G_1$ con i costi di budget

La seconda versione del nostro Algoritmo è un miglioramento della versione precedente che funziona quando non sono considerati costi di beaconing. Qui $G_2$ si ottiene normalizzando $G_1$ con il costo di budget che introdurrà un quanto di tempo aggiuntivo. In altre parole, $\delta_c$ rappresenterà un rapporto tra costi e benefici. Partendo dal presupposto che non sono presenti costi di beaconing e che abbiamo a che fare con le politiche a soglia, ogni trasmissione ha un costo indipendente e il budget speso da una politica $S$ è dato da:

$$\psi(S) = \sum_{(c,k) \in S} N_c e^{-\lambda_c \Delta(k-1)}(1 - e^{-\lambda_c \Delta})$$

e, di conseguenza,

$$G_2(\boldsymbol{\mu}) = \frac{G_1(\boldsymbol{\mu})}{\psi(\{(c, h_c + 1)\})}$$

Se modifichiamo la regola (4.5) normalizzando la funzione obiettivo attraverso il costo di budget per ogni elemento candidato, possiamo ancora mostrare l'equivalenza tra la nuova regola e l'Algoritmo 2 (versione 2). Di conseguenza, si può ricorrere di nuovo al risultato presentato in [22] e fornire un bound di qualità sulla soluzione ottenuta con la combinazione delle due versioni dell'Algoritmo 2 quando i costi beaconing non sono considerati.

**Teorema 4.2.6.** *Se non sono presenti costi di beaconing, allora si ha che*

$$\max\{F_D(K, \hat{S}^1), F_D(K, \hat{S}^2)\} \geq \frac{1}{2}(1 - \frac{1}{e}) \max_{S \subseteq \Xi \psi(S) \leq \Psi} F_D(K, S)$$

*Dimostrazione.* La dimostrazione segue immediatamente dalle considerazioni fatte precedentemente e da un adattamento lineare dei risultati presentati in [22]. □

Riportiamo in Appendice A.3 il codice MATLAB che abbiamo utilizzato per questo algoritmo (le funzioni non riportate sono le stesse utilizzate per la versione 1 dell'algoritmo).

### 4.2.3 Esempio di esecuzione

Mostriamo ora un esempio in cui è possibile vedere le differenze di funzionamento delle due versioni dell'algoritmo presentate sopra.

Prendiamo in considerazione una istanza con 3 classi e fissiamo $\tau = 25$, $\Delta = 10$, $\epsilon = 0.5$, $\Psi = 0.3924$, $L = 500$, $N_1 = 20$, $N_2 = 20$, $N_3 = 20$, $\lambda_1 = 0.001046$, $\lambda_2 = 0.001255$, $\lambda_3 = 0.012553$.
Come si può notare nelle figure 4.1 e 4.2, le due versioni dell'algoritmo portano ad un risultato diverso: infatti nella versione 1 dell'algoritmo vengono assegnati tutti i quanti di tempo disponibili alla classe con la

tecnologia WiFi e poi il budget disponibile viene esaurito dalla classe con lo ZigBee mentre la classe con il Bluetooth 4.0 non trasmette mai; nella versione 2 si assegnano come prima tutti i quanti di tempo disponibili alla classe con il WiFi, poi vengono selezionati due quanti di tempo per la classe con il Bluetooth 4.0 per poi esaurire il budget residuo con la classe rimanente (ZigBee). Questo porta ad una differente probabilità di consegna finale ($F_1 = 0.673389$, $F_2 = 0.673840$) in cui notiamo una migliore scelta delle politiche per la seconda versione dell'algoritmo (come verrà poi confermato nel Capitolo 5).

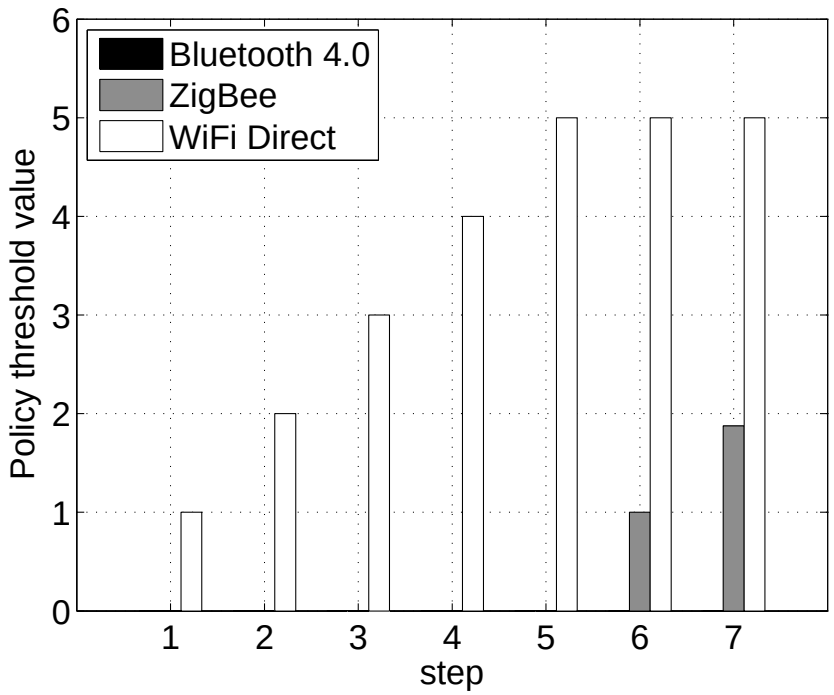


Figura 4.1: Evoluzione delle soglie di trasmissione di ogni classe ad ogni step dell'algoritmo greedy versione 1

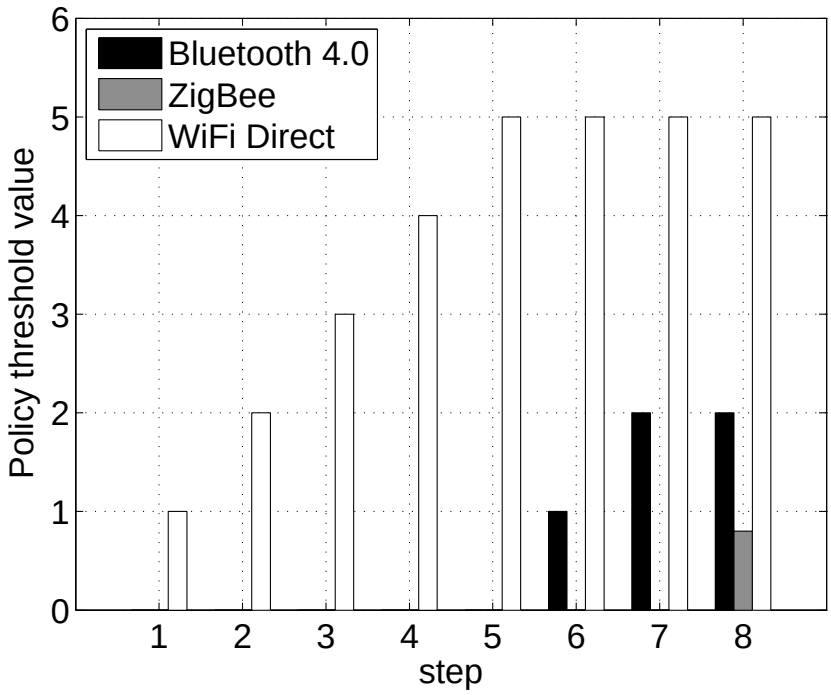


Figura 4.2: Evoluzione delle soglie di trasmissione di ogni classe ad ogni step dell'algoritmo greedy versione 2

# Capitolo 5

# Valutazioni sperimentali

In questo capitolo, forniamo alcune valutazioni sperimentali degli algoritmi proposti sopra. I risultati sono ottenuti mediante simulazioni con MATLAB con lo scopo di mostrare la correttezza del nostro approccio e valutando le prestazioni in termini di qualità della soluzione. Discuteremo inoltre alcuni problemi a livello qualitativo osservati nelle politiche che abbiamo ottenuto.

## 5.1 Parametri sperimentali

Nei nostri esperimenti, abbiamo generato delle istanze considerando un insieme finito di valori per ogni parametro del modello, vedi la Tabella 5.1 per una completa visione dei parametri. In particolare, ci concentriamo su tre diversi profili di mobilità e tre diverse tecnologie di trasmissione per i nodi mobili. I profili di mobilità sono caratterizzati da una velocità scalare media crescente. Lo scenario di riferimento è popolato da dispositivi mobili divisi in utenti a piedi, utenti in bicicletta e utenti dotati di veicolo.
Consideriamo tecnologie di trasmissione con un range di comunicazione crescente: ZigBee, Bluetooth 4.0. e Wi-Fi Direct. Deriviamo quindi i corrispondenti valori di $\rho$ (costo di una singola trasmissione) e $\beta$ (costo di

beaconing) considerando le specifiche tecniche di ogni tecnologia (range, rate, packet size, corrente, tensione, beacon), e assumendo che la dimensione di un pacchetto da inviare sia di 5kB e che il quanto di tempo sia $\Delta = 10s$.

| *deadline temporale per la consegna* ($\tau$) |
|---|
| 25, 50, 100, 250 |
| *raggio dell'ambiente circolare* ($L$) |
| 350, 500, 750,1000 |
| *numero di nodi per classe* $N_c$ |
| 9, 15, 20 |
| *profili di mobilità della classe (velocità* $v_c$*)* |
| pedoni ($1.5m/s$)<br>biciclette ($6m/s$)<br>veicoli ($9m/s$) |
| *tecnologie di trasmissione (range* $R$*)* |
| ZigBee ($R = 15m$)<br>Bluetooth 4.0 ($R = 50m$)<br>Wi-Fi Direct ($R = 100m$) |

Tabella 5.1: Parametri usati negli esperimenti

Per semplicità, assegniamo lo stesso numero di utenti (cioè di nodi) ad ogni classe $c$ e mostriamo i risultati nel caso in cui $t_c$ è non minore di $\tau$. Inoltre, la discretizzazione temporale che abbiamo adottato è $\epsilon \in \{1, 1/3, 1/5\}$. Le ragioni di questa scelta possono essere descritte in modo intuitivo attraverso i due grafici nella Figura 5.1, dove è rappresentato il lower bound teorico dato dal Teorema 4.1.2 rispetto ad una differente discretizzazione e al numero di classi. Come si può vedere, la massima discretizzazione $\epsilon = 1/5$ rappresenta una scelta ragionevole per garantire circa il 97% della qualità della soluzione ottima senza un eccesivo carico del

numero di quanti di tempo. Dall'altra parte, avendo un numero massimo di tre classi nella rete, otteniamo una situazione abbastanza vicina al caso pessimo (derivato per un numero infinito di classi) e che è computabile con il nostro algoritmo Grid 1 (ricordiamo che questo algoritmo richiede un tempo di computazione esponenziale nel numero di classi $|C|$).

## 5.2 Benchmarks

Mettiamo a confronto le prestazioni dei nostri algoritmi rispetto alle prestazioni di due algoritmi euristici facilmente computabili e di un limite superiore (UB) rispetto al valore della politica ottima.

### 5.2.1 Greedy sulla frequenza di arrivo (Cascata)

Questo algoritmo funziona nel seguente modo: si ordinano le classi in ordine decrescente di $\lambda_c$, quindi si assegna tutto il budget possibile alle classi dalla prima fino all'ultima secondo l'ordine precedentemente stabilito. Ad esempio, date tre classi con $\lambda_1 = 0.3, \lambda_2 = 0.2, \lambda_3 = 0.1$, l'algoritmo assegna tutto il budget possibile alla classe 1 e, se c'è ancora budget disponibile, allora tutto il budget residuo è assegnato alla classe 2 e così via. La logica è che ci aspettiamo che maggiore è la frequenza di arrivo $\lambda$, più grande è la probabilità di consegna.
La complessità di questo algoritmo è ovviamente bassa dato che la politica può essere trovata risolvendo al massimo $|C|$ equazioni.

Riportiamo in Appendice A.4 il codice MATLAB che abbiamo utilizzato per questo algoritmo.

### 5.2.2 Politiche indipendenti (una soglia uguale per tutte le classi)

Questo algoritmo cerca la soluzione ottima di un problema maggiormente vincolato in cui le politiche relative a tutte le classi sono le stesse, formal-

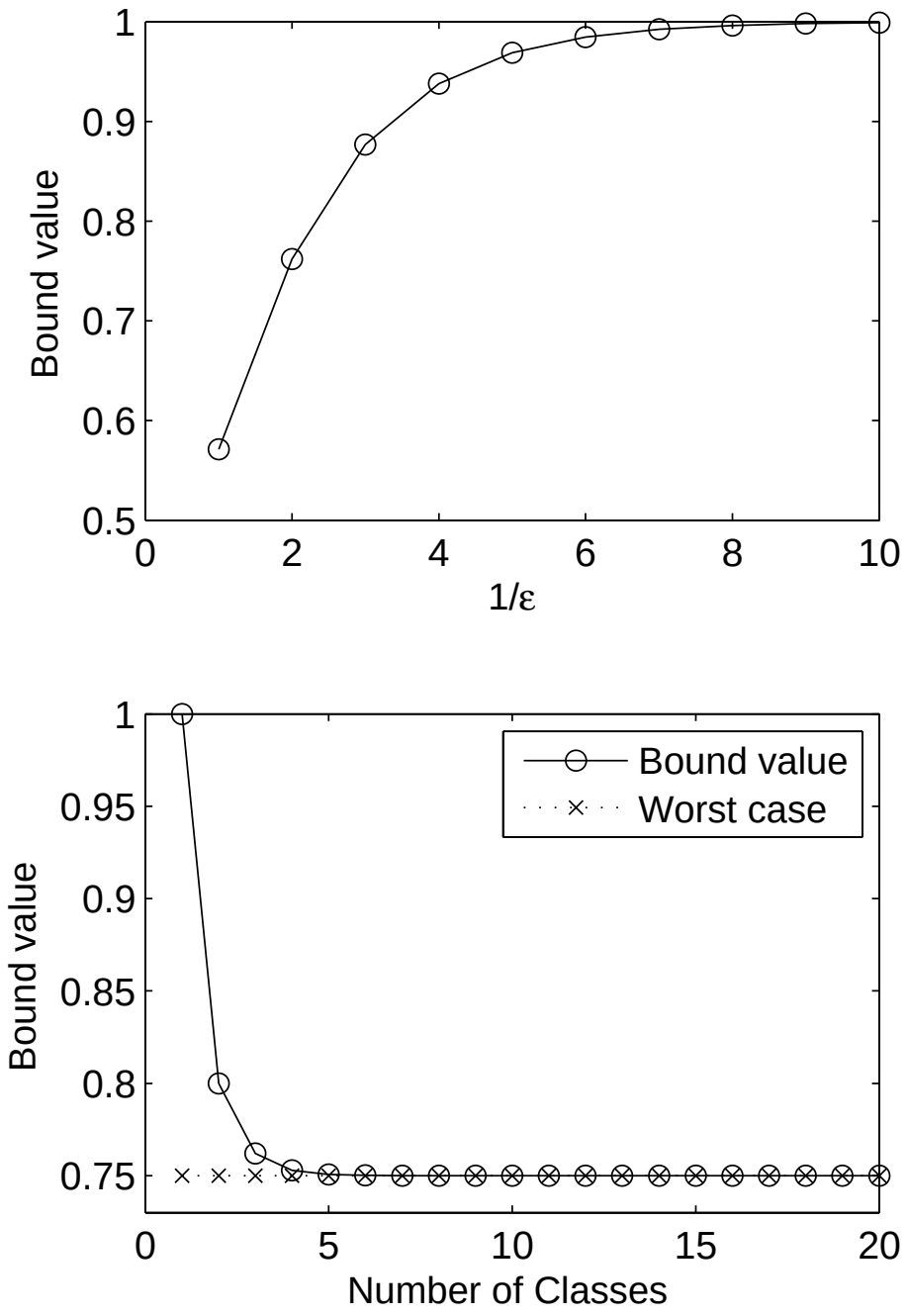


Figura 5.1: Lower bound teorico sulla qualità della soluzione (Teorema 4.1.2)

mente $\mu(k) = \mu_c(k)$ per ogni $c$, e, quando la politica è probabilistica, allora o la sorgente trasmette a tutte le classi o non trasmette affatto. Questa ultima ipotesi porta ad una nuova formulazione del vincolo di budget:

$$\sum_{c \in C} \rho_c N_c \cdot (1 - Q_{c,0,K}(\mu)) + \sum_{\omega \in \Omega} \sum_{k=0}^{K-1} \beta_\omega \cdot \mu(k) \leq \Psi$$

Dalla Proprietà 3.3.1, la politica ottima è tale che il budget $\Psi$ sia completamente consumato e quindi la disuguaglianza sopra vale con l'uguaglianza. Pertanto, il problema di ottimizzazione si riduce al problema di trovare la politica che consuma completamente il budget. Formalmente, interpretando la soglia $h$ (indipendente dalla classe) come una variabile continua, possiamo scrivere:

$$g(h) = \sum_{c \in C} N_c \cdot e^{-\lambda_c \Delta h} - \sum_{\omega \in \Omega} \left( \sum_{c \in C_\omega} N_c - \frac{\beta_\omega}{\rho_c} \cdot h + \frac{\Psi}{\rho_c} \right) = 0$$

La funzione $g$ è una funzione ad una singola variabile strettamente monotona decrescente in $h$ infinitamente differenziabile. Tale funzione ammette un solo zero, e quindi la suddetta equazione ammette una sola soluzione. Tale soluzione può essere trovata (in modo approssimato) utilizzando il metodo di Newton che, in questo caso, a causa della proprietà della funzione, ha una velocità di convergenza quadratica (il numero di cifre corrette approssimativamente raddoppia ad ogni passo). Così, si ottiene una soluzione approssimata di alta qualità in tempi molto brevi.

Riportiamo in Appendice A.5 il codice MATLAB che abbiamo utilizzato per questo algoritmo.

### 5.2.3 Esempio di esecuzione

In questa parte vogliamo mostrare come si comportano i due algoritmi presentati sopra rispetto all'algoritmo greedy. Consideriamo la stessa istanza utilizzata in 4.2.3 con la sola differenza che settiamo $\epsilon = 1$ (ricordiamo che

gli altri parametri sono $\tau = 25$, $\Delta = 10$, $\Psi = 0.3924$, $L = 500$, $N_1 = 20$, $N_2 = 20$, $N_3 = 20$, $\lambda_1 = 0.001046$, $\lambda_2 = 0.001255$, $\lambda_3 = 0.012553$).
In Figura 5.2 è possibile vedere il funzionamento degli algoritmi rispetto al greedy (versione 1). Come si poteva facilmente intuire, l'algoritmo a Cascata punta sulle classi che possiedono una tecnologia di trasmissione veloce fino ad esaurimento del budget, mentre l'algoritmo delle politiche indipendenti trova una soglia uguale per ogni classe. Ovviamente queste politiche non portano ad un risultato ottimo in quanto le politiche trovate sono ovviamente diverse da quelle trovate dall'algoritmo Greedy. Più avanti si mostreranno altri esempi in cui viene confermata questa tendenza. C'è comunque da notare come il tempo di esecuzione di questi algoritmi sia trascurabile non solo rispetto all'algoritmo Grid, ma anche rispetto all'algoritmo Greedy.

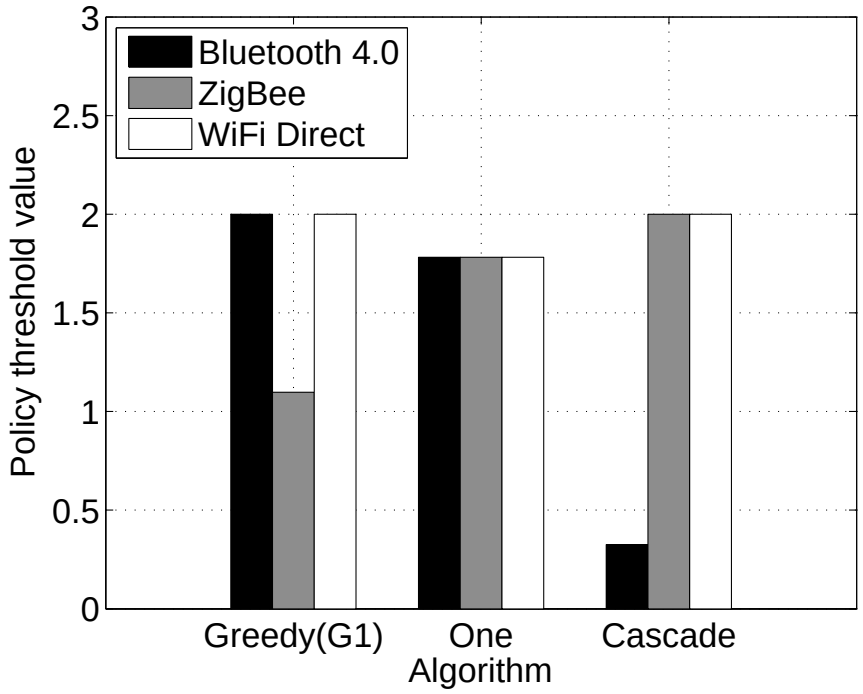


Figura 5.2: Soglie delle politiche di tre classi ottenute con gli algoritmi Greedy (versione 1), Politiche Indipendenti e Cascata

### 5.2.4 Upper Bound sul valore ottimo (UB)

Un upper bound sul valore della soluzione ottima può essere trovato mediante una variazione dell'algoritmo descritto nella Sezione 4.1. Più precisamente, si usa l'Algoritmo 1 per enumerare tutte le politiche che consumano interamente il budget e cambiamo ogni politica arrotondando ogni $h_c$ al più piccolo intero superiore e aggiungendo 1 ad ogni classe $c$ con politica intera. Si noti che queste nuove politiche violano, ovviamente, il vincolo di budget. Tra tutte queste politiche cerchiamo quella che massimizza la probabilità di consegna. Il suo valore è un limite superiore del valore della politica ottima. Nei grafici, indichiamo questo valore con $UB$.

Segue la dimostrazione. Chiamo $\boldsymbol{\mu}^*$ la politica ottima con le soglie $h_c^*$ (potenzialmente frazionarie). Chiamo $\hat{\boldsymbol{\mu}}$ una generica politica ottenuta come descritto sopra. Si può facilmente osservare che esiste sempre una politica $\hat{\boldsymbol{\mu}}$ tale che $\hat{h}_c \geq h_c^*$ per ogni classe $c$ (segue dal fatto che, fissate le politiche di tutte le classi tranne una, la politica della classe restante che consuma interamente il bilancio è sempre una). Pertanto, dato che la funzione obiettivo è strettamente monotona in $h_c$, il valore obiettivo di $\hat{\boldsymbol{\mu}}$ è strettamente migliore rispetto al valore $\boldsymbol{\mu}^*$.

Riportiamo in Appendice A.6 il codice MATLAB che abbiamo utilizzato per questo algoritmo.

## 5.3 Risultati sperimentali

La Figura 5.3 riporta come $F_D/UB$ varia al variare del valore dei parametri $\tau, L, N_c$ come indicato in Tabella 5.1, $|C| \in \{1, 2, 3\}$, e $\frac{1}{\epsilon} = 5$. (L'Algoritmo 2 che utilizza la funzione $G_i$ è indicato con greedy construction $(i)$). Per ogni parametro, mediamo $F_D/UB$ su tutte le altre istanze che condividono lo stesso valore per quel parametro. Si può osservare che con la ricerca grid e greedy si ottiene una migliore prestazione in ogni caso quando comparate con gli algoritmi di benchmark basati sulla frequenza di arrivi (Cascata) e sulle politiche indipendenti dalle classi (vale a dire una

politica uguale per ogni classe $c$). Infatti questi algoritmi, non sfruttando la conoscenza dei parametri delle diverse classi e considerando esclusivamente la frequenza di arrivo, hanno dimostrato di raggiungere prestazioni molto simili tra loro. Incrementando il valore di $\tau$, si può vedere come questo gap con i benchmark si riduce, suggerendo l'intuizione che quando la deadline per la consegna di un pacchetto è alta abbastanza, anche con questi semplici algoritmi è possibile ottenere una buona probabilità di consegna.
Un altro aspetto che può essere osservato è che gli algoritmi greedy si sono rivelati molto efficaci nei casi esaminati, dato che essi sono stati in grado di ottenere prestazioni elevate comparabili all'algoritmo grid. Incrementando il valore di $L$, si può notare come il gap con i benchmark aumenti, invece il gap continua ad essere approssimativamente costante al variare di $N_c$ e $C$. Un fatto interessante è che il rapporto di approssimazione dei nostri algoritmi è costante (ad esempio, $> 99\%$) rispetto a tutti i valori dei parametri.

Una panoramica più dettagliata su come le prestazioni cambiano col variare di $\tau$ è mostrata mediante i boxplot in Figura 5.4 e in Figura 5.5. Questi grafici mostrano la similarità delle prestazioni tra gli algoritmi grid e greedy. Queste ultime hanno ottenuto prestazioni peggiori per un limitato numero di istanze. Ed è anche evidente il fatto che avendo una discretizzazione del tempo più fine migliora notevolmente la qualità della soluzione.

I risultati suggeriscono che l'Algoritmo greedy sembra essere un approccio abbastanza efficace per approssimare la politica ottima richiedendo, allo stesso tempo, molto meno sforzo computazionale rispetto all'Algoritmo grid. Nella Figura 5.6, mostriamo un confronto tra il tempo di computazione ottenuto mediante gli algoritmi di ricerca grid e greedy rispettivamente. In particolare, abbiamo valutato la scalabilità degli algoritmi al crescere del numero di classi di nodi presenti nella rete. Per ottenere questi risultati, abbiamo fissato i valori di alcuni parametri ($\epsilon = 1/3$, $\tau = 100$,

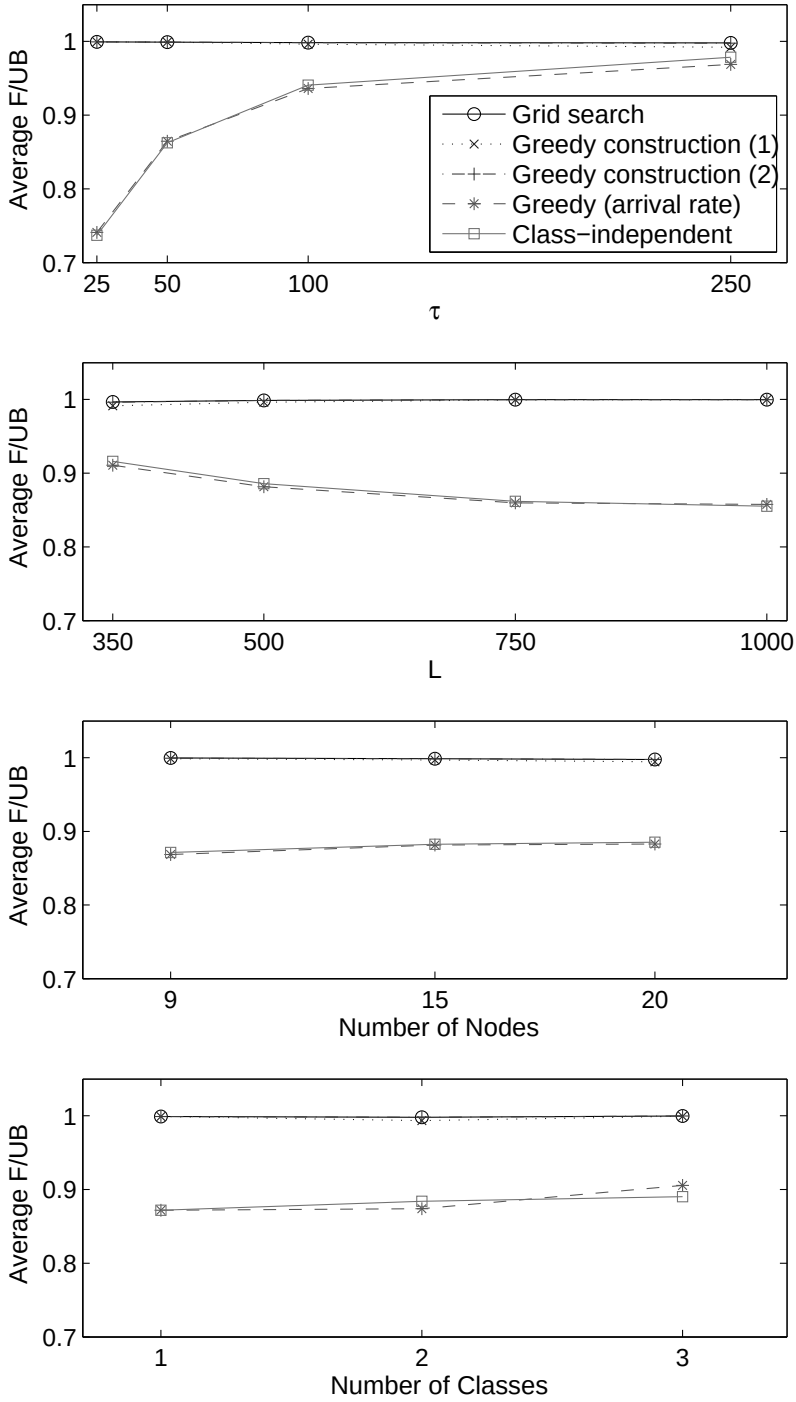


Figura 5.3: Media di $F_D/UB$ rispetto ai diversi parametri con $\frac{1}{\epsilon} = 5$

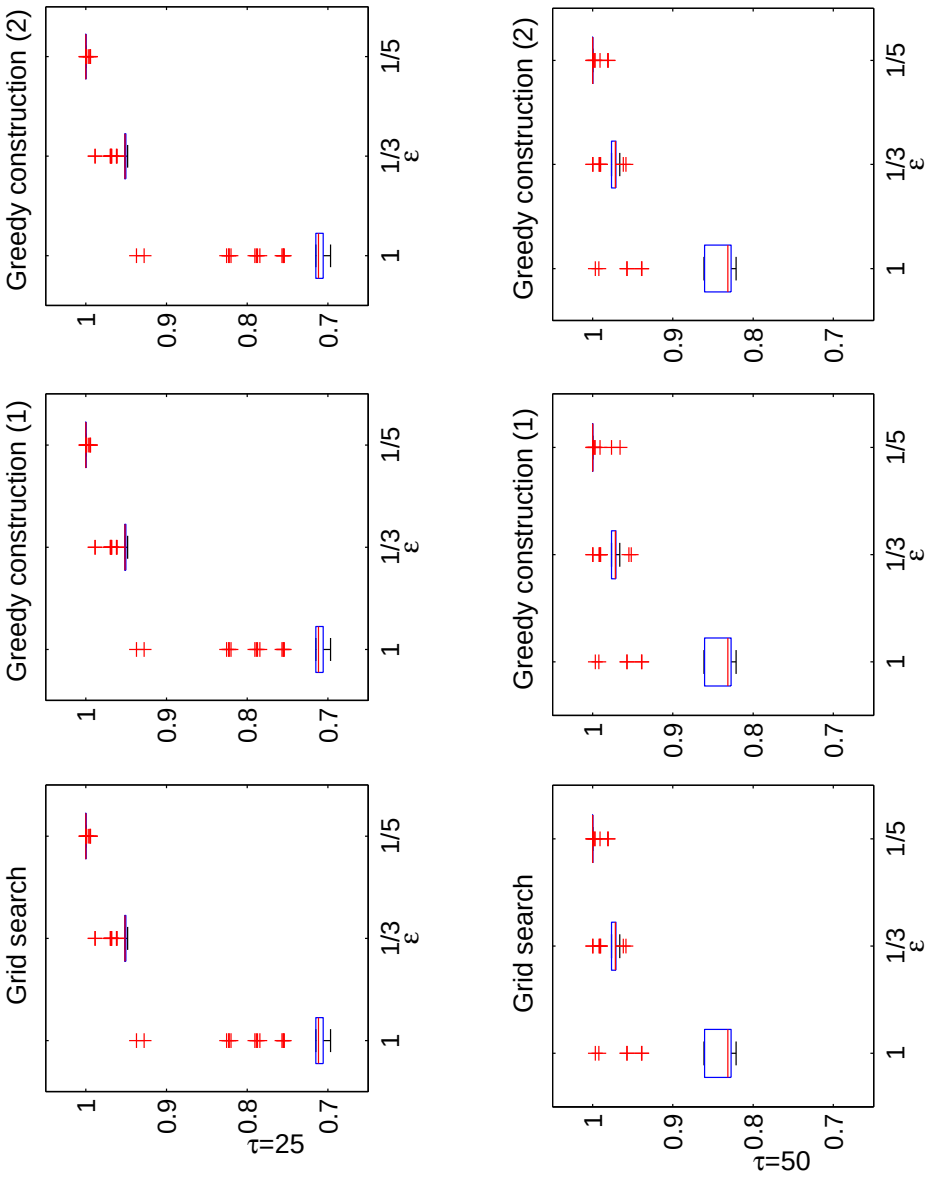


Figura 5.4: Boxplot di $F_D/UB$ rispetto a $\tau = \{25, 50\}$ per i diversi algoritmi

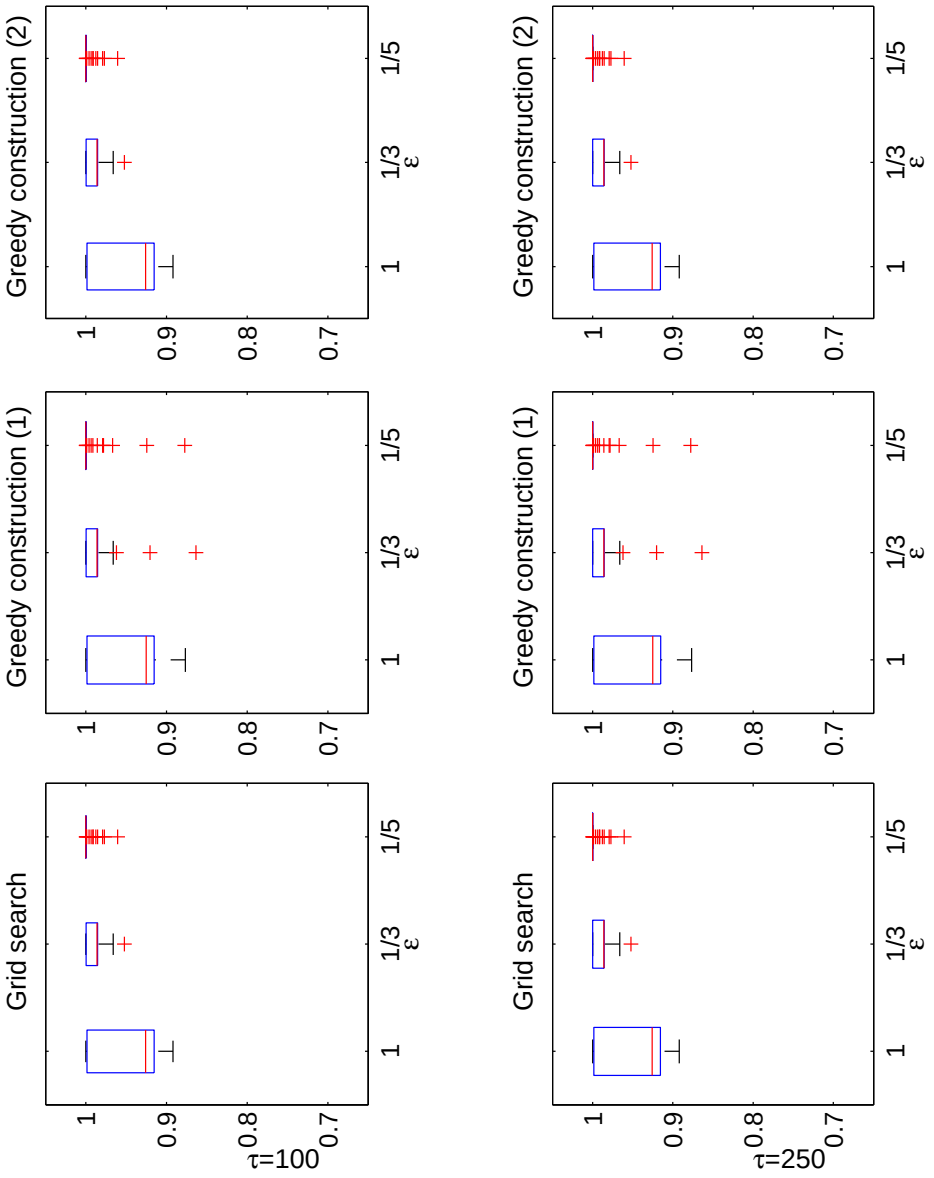


Figura 5.5: Boxplot di $F_D/UB$ rispetto a $\tau = \{100, 250\}$ per i diversi algoritmi

$N_c = 10$, $L = 500$) e abbiamo generato dei profili di mobilità e tecnologie di trasmissione casuali con distribuzione uniforme nei seguenti intervalli: $R_c \in [15, 50]$, , $v_c \in [1, 15]$ $\rho_i \in [0.05, 0.25]$ $\beta_c \in [3 \times 10^{-7}, 8 \times 10^{-7}]$. Si può facilmente vedere come la ricerca grid mostri una crescita esponenziale nel tempo, mentre la costruzione greedy si dimostra molto efficiente anche per un numero di classi elevato. Considerando una deadline di un'ora, la ricerca grid non è stata in grado di calcolare la soluzione del problema con più di 4 classi, mentre la costruzione greedy è riuscita a calcolare la soluzione fino a 800 classi (Figura 5.7).

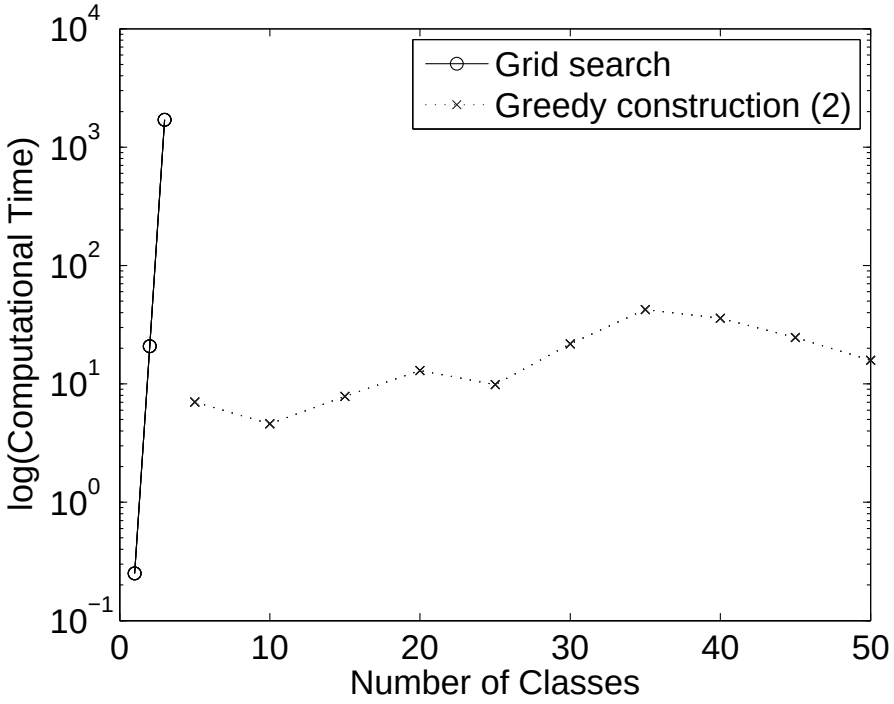


Figura 5.6: Scalabilità del tempo (in secondi) con il numero di classi

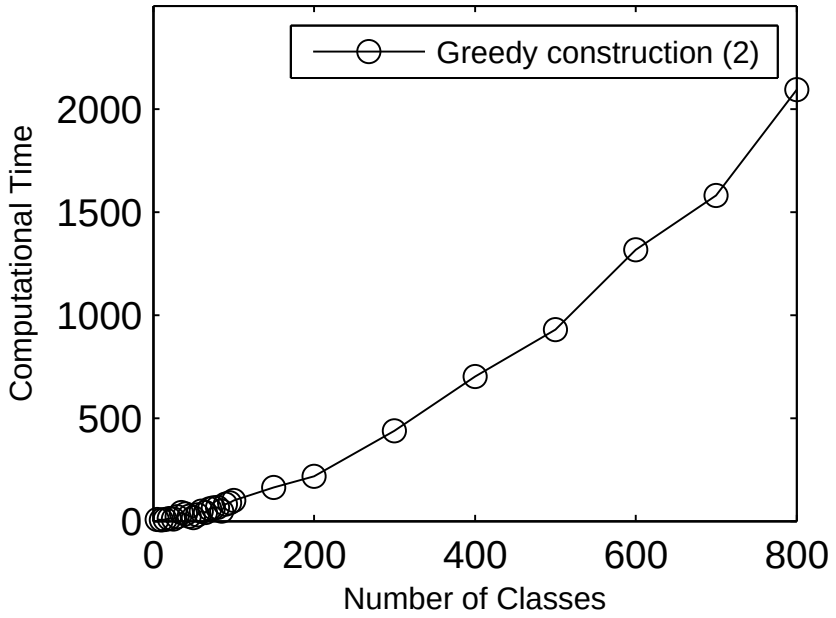


Figura 5.7: Scalabilità del tempo con il numero di classi per Greedy($G_2$)

In Figura 5.8 viene mostrato come il costo di beaconing incide sul calcolo della probabilità di consegna finale. Per fare ciò, abbiamo utilizzato l'algoritmo greedy (versione 2) con la massima discretizzazione prevista (ovvero $\epsilon = 1/5$) ed abbiamo fatto variare il valore di $\beta$; come si può vedere è necessario aumentare di parecchio il valore di $\beta$ affinché si abbia una rilevanza sul calcolo della probabilità $F_D$.

In Figura 5.9 è rappresentata una valutazione qualitativa delle politiche trovate dai nostri algoritmi. Consideriamo un valore di riferimento per il budget $\Psi$ e mostriamo come le soglie delle politiche ottime (ottenute mediante ricerca grid) sono distribuite sulle tre diverse tecnologie utilizzate. Si può osservare come, all'aumentare del budget, la politica ottima tende a schedulare trasmissioni con tutte e tre le tecnologie. Quando il budget diminuisce, allora la politica ottima cerca di trasmettere maggiormente su quella tecnologia che ha un range di comunicazione più elevato rispetto alle altre.

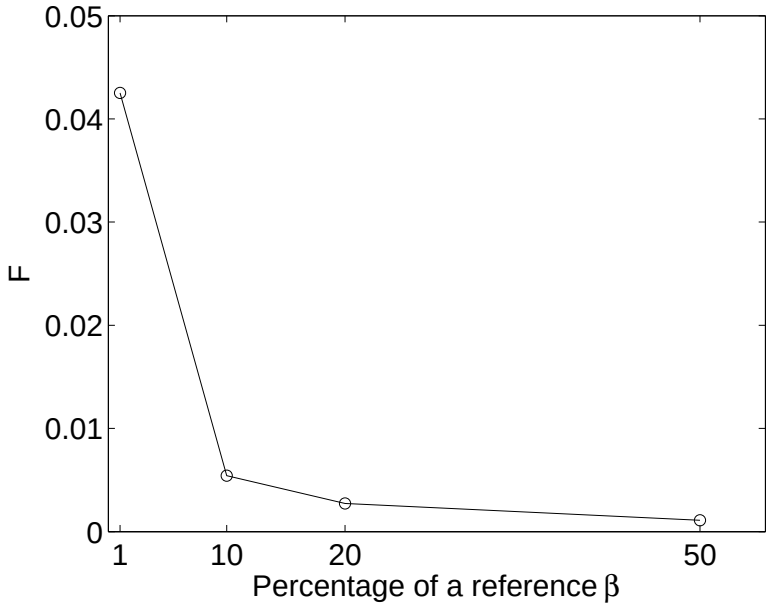


Figura 5.8: Andamento di $F_D$ al variare di $\beta$ con Greedy(2)

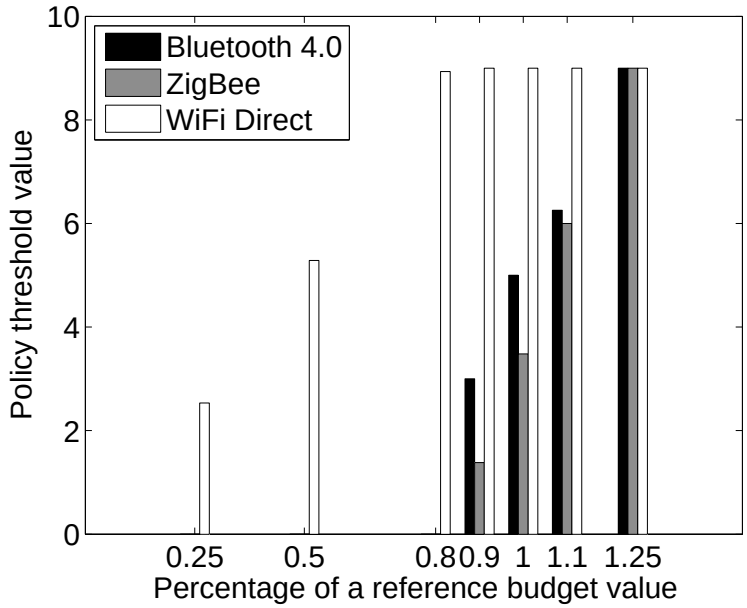


Figura 5.9: Politiche con diversi valori del budget $\Psi$

Infine, in Figura 5.10 si valutano ancora qualitativamente le politiche al variare di $\tau$. A conferma delle osservazioni precedenti, in presenza di valori piccoli di $\tau$ le politiche sono distribuite su tutte le tecnologie in quanto non c'è abbastanza tempo per consumare interamente il budget. Col crescere di $\tau$, la politica ottima trasmette prima sulla tecnologia che ha il range di comunicazione più elevato per poi spostarsi sulle altre quando viene raggiunto l'ultimo istante di tempo utile per la trasmissione. Infatti, quando $\tau = 250$ la politica ottima è trasmettere solo con WiFi in quanto la deadline è abbastanza lontana da trasmettere solo su una tecnologia ed esaurire tutto il budget disponibile.

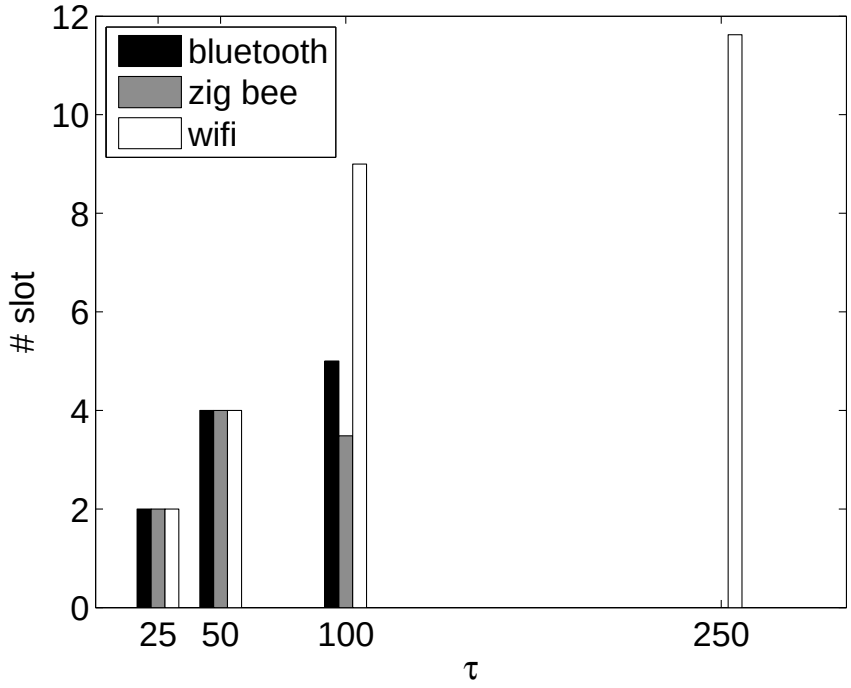


Figura 5.10: Politiche con diversi valori di $\tau$

Abbiamo comparato il nostro approccio con la tecnica di approssimazione fluida proposta in [28] assumendo costi di beaconing nulli e aumentando il numero di nodi per classe da 1 a 200. Abbiamo osservato che,

come con altre applicazioni di approssimazione fluida, qui l'errore (misurato come l'inefficienza della soluzione rispetto al nostro algoritmo di costruzione greedy (versione 2)) causa di approssimazione fluida diventa trascurabile ($< 1\%$) per più di 100 utenti, mentre è significativo ($10-20\%$) negli altri casi. In aggiunta a ciò, fissato il numero di classi, i nostri algoritmi di costruzione greedy sono stati in grado di richiedere tempi di computazione notevolmente inferiori rispetto all'approccio con approssimazione fluida. Come risultato, i nostri algoritmi producono soluzioni più efficienti con un numero piccolo di nodi rispetto all'approssimazione fluida e richiedono un tempo computazionale molto minore per tutti i parametri, essendo pertanto applicabili con un grande numero di classi.

# Capitolo 6

# Conclusioni

In questo lavoro abbiamo studiato il routing a due hop per le Delay Tolerant Networks quando sono presenti tecnologie di trasmissione eterogenee e tenendo in considerazione i costi di beaconing e le deadline oltre le quali i nodi scartano il pacchetto.
A differenza della letteratura, che comunemente ricorre all'approssimazione fluida per trovare le politiche ottime (fornendo una soluzione esatta solo quando il numero di nodi è infinito, ma una approssimazione grossolana negli altri casi), noi adottiamo un approccio di ricerca operativa, formulando il problema come un problema di ottimizzazione e progettando schemi di approssimazione con bound teorici. Il risultato cruciale è che il problema è sub-modulare e quindi l'ottimizzazione greedy porta ad una soluzione di alta qualità con garanzie teoriche.
In particolare, l'algoritmo greedy, che massimizza ad ogni iterazione il guadagno marginale normalizzato con i costi marginali, ha fornito in tutti gli esperimenti effettuati un rapporto di approssimazione maggiore del 99% indipendentemente dal numero di utenti e classi nella rete. Inoltre, la strategia proposta scala linearmente al crescere del valore dei parametri. La robustezza del nostro algoritmo è confermata dalla deviazione standard estremamente bassa dei risultati sperimenatli. L'algoritmo greedy può quindi essere applicato a qualsiasi istanza (da quelle più semplici a

quelle più complesse in termini di numero di classi e numero di nodi) ottenendo in ogni caso soluzioni di alta qualità.
Questo lavoro ha portato alla scrittura e alla sottomissione dell'articolo [12].

Due problematiche interessanti che restano aperte in questo campo e che seguono in modo pertinente ai contributi di questa tesi sono:

- la progettazione di tecniche di apprendimento online ottime che minimizzino l'errore quando i parametri del problema (frequenza di contatto tra nodi, costi di beaconing e tecnologie di trasmissione) non sono noti a priori;
- la formalizzazione del problema con routing a due hop nel caso in cui tutti i nodi, inclusa la sorgente e la destinazione, possono utilizzare più tecnologie simultaneamente.

# Bibliografia


[1] E. Altman. Competition and cooperation between nodes in delay tolerant networks with two hop routing. *Network Control and Optimization*, pages 264–278, 2009.

[2] E. Altman, A. P. Azad, T. Basar, and F. De Pellegrini. Optimal activation and transmission control in delay tolerant networks. In *INFOCOM*, pages 106–110, 2010.

[3] E. Altman, A. Prakash Azad, T. Basar, and F. De Pellegrini. Combined optimal control of activation and transmission in delay-tolerant networks. *IEEE ACM T NETWORK*, 21(2):482–494, 2013.

[4] E. Altman, T. Basar, and F. De Pellegrini. Optimal control in two–hop relay routing. In *CDC*, pages 2729–2735, 2009.

[5] E. Altman, T. Basar, and F. De Pellegrini. Optimal control in two–hop relay routing. *IEEE T AUTOMAT CONTR*, 56(3):670–675, 2011.

[6] E. Altman, G. Neglia, F. De Pellegrini, and D. Miorandi. Decentralized stochastic control of delay tolerant networks. In *INFOCOM*, pages 1134–1142. IEEE, 2009.

[7] E. Altman, G. Neglia, F. De Pellegrini, and D. Miorandi. Decentralized stochastic control of delay tolerant networks. In *INFOCOM*, pages 1134–1142, 2009.

[8] E. Altman and F. De Pellegrini. Forward correction and fountain codes in delay–tolerant networks. *IEEE ACM T NETWORK*, 19(1):1–13, 2011.

[9] E. Altman, L. Sassatelli, and F. De Pellegrini. Dynamic control of coding for progressive packet arrivals in DTNs. *IEEE T WIREL COMMUN*, 12(2):725–735, 2013.

[10] R. E. Azouzi, F. De Pellegrini, H. B. A. Sidi, and V. Kamble. Evolutionary forwarding games in delay tolerant networks: Equilibria, mechanism design and stochastic approximation. *COMPUT NETW*, 57(4):1003–1018, 2013.

[11] N. Banerjee, M. D. Corner, and B. Neil Levine. Design and field experimentation of an energy–efficient architecture for dtn throwboxes. *IEEE ACM T NETWORK*, 18(2):554–567, 2010.

[12] N. Basilico, N. Gatti, L. Goffredi, and M. Cesana. Beaconing-aware optimal policies for two-hop routing in multi-class delay tolerant networks. *sottomesso a INFOCOM*, 2013.

[13] W. Chahin, R. El Azouzi, F. De Pellegrini, and A. P. Azad. Blind online optimal forwarding in heterogeneous delay tolerant networks. In *Wireless Days*, pages 1–6, 2011.

[14] B.B. Chen and M.C. Chan. Mobicent: a credit-based incentive system for disruption tolerant network. In *INFOCOM*, pages 1–9, 2010.

[15] R. El-Azouzi, F. De Pellegrini, and V. Kamble. Evolutionary forwarding games in delay tolerant networks. In *WiOpt*, pages 76–84, 2010.

[16] V. Goncharov. Delay-tolerant networks. 2010.

[17] R. Groenevelt, P. Nain, and G. Koole. The message delay in mobile ad hoc networks. *PERFORM EVALUATION*, 62(1–4):210–228, 2005.

[18] M. Grossglauser and D. N. C. Tse. Mobility increases the capacity of ad hoc wireless networks. *IEEE ACM T NETWORK*, 10(4):477–486, 2002.

[19] A. Guerrieri, I. Carreras, F. De Pellegrini, D. Miorandi, and A. Montresor. Distributed estimation of global parameters in delay–tolerant networks. *COMPUT COMMUN*, 33(13):1472–1482, 2010.

[20] S. Jain, K. Fall, and R. Patra. *Routing in a delay tolerant network*, volume 34. ACM, 2004.

[21] K.H. Kabir, M. Sasabe, and T. Takine. Design and cof self-organized data aggregation using evolutionary game theory in delay tolerant networks. In *WoWMoM 2009*, pages 1–6, 2009.

[22] A. Krause and D. Golovin. Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems (to appear)*. Cambridge University Press, 2012.

[23] A. Krifa, C. Baraka, and T. Spyropoulos. Optimal buffer management policies for delay tolerant networks. In *SECON*, pages 260–268, 2008.

[24] Q. Li, S. Zhu, and G. Cao. Routing in socially selfish delay tolerant networks. In *INFOCOM*, pages 1–9, 2010.

[25] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions - i. *Mathematical Programming*, 14(1):265–294, 1978.

[26] T. Ning, Z. Yang, X. Xie, and H. Wu. Incentive-aware data dissemination in delay-tolerant mobile networks. SECON, 2011.

[27] D. Niyato, P. Wang, W. Saad, and A. Hjorungnes. Coalition formation games for improving data delivery in delay tolerant networks. In *GLOBECOM*, pages 1–5, 2010.

[28] F. De Pellegrini, E. Altman, and T. Basar. Optimal monotone forwarding policies in delay tolerant mobile ad hoc networks with multiple classes of nodes. In *WiOpt*, pages 497–504, 2010.

[29] O. Perry and W. Whitt. A fluid approximation for service systems responding to unexpected overloads. *OPER RES*, 59(5):1159–1170, 2011.

[30] R. S. Randhawa. Accuracy of fluid approximations for queueing systems with congestion-sensitive demand and implications for capacity sizing. *OPER RES LETT*, 41(1):27–31, 2013.

[31] H. Tembine, E. Altman, R. El Azouzi, and Y. Hayel. Evolutionary games in wireless networks. *IEEE T SYST MAN CY B*, 40(3):634–646, 2010.

[32] F. Warthman et al. Delay-tolerant networks (dtns): A tutorial, 2003.

[33] H. Zhu, X. Lin, R. Lu, Y. Fan, and X. Shen. Smart: A secure multilayer credit-based incentive scheme for delay-tolerant networks. *IEEE T VEH TECHNOL*, 58(8):4628–4639, 2009.

# Appendice A

# Codice Matlab

## A.1 Algoritmo Grid

grid.m

```
-----------------------------------------
function [a policy]= grid_multiclass(pa,eps)
dt = pa.Delta;
kk = pa.K;
be_cost = pa.beta;

for e = 1:size(eps,2)
    pa.Delta = eps(e)*dt;
    pa.K = kk*(1/(eps(e)));
    pa.beta = eps(e)*be_cost;
    global F
    global B
    global c_f
    global pol
    global ord
    pol = NaN(1,pa.C);
    F = 0;
```

```
    B = NaN(pa.C-1,pa.K);

    for f = 1:pa.C
        ord = 1:pa.C;
        c_f = f;
        ord(f) = [];
        path = NaN(1,pa.C-1);
        F_hat = VALUTA(pa, path);
    end

    FF(e,1) = F;
    pol_ott(e,:) = pol;
end

a = FF;
policy = pol_ott;
```

VALUTA.m

```
---------------------------------
function F_hat = VALUTA(pa, path)
global F
global pol
global B
global c_f
global ord

if all(~isnan(path))
    mu_scal = zeros(pa.C,1);
    for i = 1 : size(path,2)
```

```
        mu_scal(ord(i),1) = B(i, path(i));
    end
    mu_vec = scal_to_vec(pa,mu_scal);
    mu_scal(c_f,1)=min(mu_hat_multi_tech(c_f,mu_vec,pa),pa.K-1);
    mu_vec = scal_to_vec(pa,mu_scal);
    F_hat = FD(pa,mu_vec);
    if F_hat > F
            F = F_hat;
            for i = 1 : pa.C
                pol(i) = mu_scal(i);
            end
     end
else
    n = isnan(path);
    y = find (n,1,'first');
    V = Expand_v2(pa, path, y);

    if V <= 0
        disp('Errore: L''intervallo ha 0 elementi.')
    end
        for w = 1:V
        new_path = [path(1:y-1), w, path(y+1:end)];
        F_hat = VALUTA(pa, new_path);
    end
end
```

Expand_v2.m

```
-----------------------------------
function V = Expand_v2(pa, path, y)
```

```
global B
global ord
B(y,:) = NaN(1,pa.K);

mu_scal_1 = zeros(pa.C,1);
for i = 1 : y - 1
    mu_scal_1(ord(i),1) = B(i, path(i));
end
mu_vec_1 = scal_to_vec(pa,mu_scal_1);
r_1 = min(mu_hat_multi_tech(ord(y),mu_vec_1,pa),pa.K-1);
if r_1 < 0
    r_1 = 0;
end

mu_scal_2 = ones(pa.C,1)*(pa.K-1);
for i = 1 : y - 1
    mu_scal_2(ord(i),1) = B(i, path(i));
end
mu_scal_2(ord(y),1) = 0;
mu_vec_2 = scal_to_vec(pa,mu_scal_2);
r_2 = min(mu_hat_multi_tech(ord(y),mu_vec_2,pa),pa.K-1);

temp_r_1 = floor(r_1);
temp_r_2 = ceil(r_2);
if temp_r_2 > temp_r_1
temp_r_2 = temp_r_1;
end

i1 = max([0, temp_r_2]);
i2 = min([pa.K-1, temp_r_1]);
i = i1:i2;
```

```
V = size(i,2);
if V <= 0
    disp('Errore: L''intervallo calcolato ha 0 elementi.')
end
I = [i NaN(1,pa.K-V)];
B(y,:) = I;
```

FD.m

```
----------------------
function r = FD(pa,mu)
F = 1;

for i = 1:pa.C
    M = zeros(pa.K, pa.N(i)+1);
    M(1,1) = 1;

    for r = 2:pa.K
        XSTAR = 0;

        for c = 1:pa.N(i)+1
            Q = exp(-pa.lambda(i) * pa.Delta * mu(i,r-1));
            P_rc = 0;

            for h = 0:c-1
                arr = c - 1 - h;
                P1 = (1-Q)^arr;
                P2 = Q^(pa.N(i)-arr);
                P3 = M(r-1,h+1);
```

```
                if arr > pa.N(i)
                    disp('errore')
                end
                L = nchoosek(pa.N(i),arr);
                P_rc = P_rc + P1 * P2 * P3 * L;
            end

            M(r,c) = P_rc;
            XSTAR = XSTAR + P_rc *
             *exp(-pa.lambda(i)*pa.Delta*(c-1));

        end

        F = F * XSTAR;
    end
end

r = 1 - F;
```

mu_hat_multi_tech.m

```
----------------------------------------
function m = mu_hat_multi_tech(i,mu,pa)
A = 0;
h_j = 0;
h_i = sum ( mu(i,:));
B = 0;

for j=1:pa.C
    if j ~= i
```

```
        h_j = sum(mu(j,:));
        A = A + (pa.N(j)*(1-exp(-pa.lambda(j)*pa.Delta*
        * h_j))) * pa.rho(j);
if pa.beta(j) == pa.beta(i)
if h_j > B
B = h_j;
           end
        end
    end
end

beta_vec = unique(pa.beta);
CB = 0;

for b = 1:size(beta_vec,2)
if pa.beta(b) ~= pa.beta(i)
Q = 0;
for j = 1:pa.C
if pa.beta(j) == pa.beta(b)
cost = sum(mu(j,:));

if cost > Q
Q = cost;
end
end
end
CB = CB + Q * pa.beta(b);
end
end

X = 1-(pa.psi/pa.rho(i)-A/pa.rho(i)-(pa.beta(i)/pa.rho(i)*
```

```
*B+CB/pa.rho(i)))/pa.N(i);

if X > 0
phi_i = log(X);
    m = (phi_i/(-pa.lambda(i)*pa.Delta))-h_i;
else
    m = pa.K-1;
end

if pa.beta(i) > 0
    if (m + h_i) > B
        pol = sym('pol');
        g = pa.rho(i)*(pa.N(i)*(1-exp(-pa.lambda(i)*
        *pa.Delta*(pol))))+pa.beta(i)*pol-pa.psi+A+CB;
        [sol,nit]=newton(g,0,0.00001,100);

        m = sol - h_i;
    end
end
```

## A.2 Agoritmo Greedy (versione 1)

greedy_opt.m

```
---------------------------------------
function [d policy] = greedy_opt(pa,eps)
dt = pa.Delta;
kk = pa.K;
be_cost = pa.beta;
global S
global H
global P

for e = 1:size(eps,2)
    pa.Delta = eps(e)*dt;
    pa.K = kk*(1/(eps(e)));
    pa.beta = eps(e)*be_cost;
    mu = zeros(pa.C,pa.K-1);
    H = zeros(1, pa.C);
    n_max = max(pa.N);
    P = NaN(pa.C, n_max + 2);
    S = NaN(pa.K, n_max+1, pa.C);

    for i=1:pa.C
        S(:,1:pa.N(i)+1,i) = 0;
        S(:,1,i) = 1;
        P(i,1:pa.N(i)+2) = [1 S(1,1:pa.N(i)+1,i)];
    end
    cc = 1;

    while (checkbudget(pa,mu)==1) && ~all(H==pa.K-1) &&
     && (cc >= 1)
        mg = zeros(1,pa.C);
```

```
fix_S = S;
bg = FD_greedy(pa, fix_S);

for i=1:pa.C

    if (H(i)+1) <= pa.K-1
        max_all = mu_hat_multi_tech(i,mu,pa);
        mu(i, H(i)+1) = min( [ 1  max_all ] );
        S_hat = S;
        T_hat = update_S(pa, mu, i);
        S_hat(:,:,i) = T_hat;
        ag = FD_greedy(pa, S_hat);
        mg(i) = ag - bg;
        mu(i,H(i)+1) = 0;
    end
end
[maxval istar] = max(mg);

if (maxval == 0) && (H(istar) + 1 <= pa.K-1)
elseif (maxval == 0) && (H(istar+1) + 1 <= pa.K-1)
    istar = istar +1;
elseif (maxval == 0) && (H(istar+2) + 1 <= pa.K-1)
    istar = istar + 2;
end

if maxval>= 0

    cc = mu_hat_multi_tech(istar,mu,pa);
    mu(istar,H(istar)+1) = min( [ 1  cc ] );
    S_new = update_S(pa,mu,istar);
    S(:,:,istar) = S_new;
```

```
            H(istar) = H(istar)+1;
        end
    end
    S_final = S;
    R(e) = FD_greedy(pa, S_final);
    MU = vec_to_scalar(pa,mu);

    for i=1:pa.C
        pol(e,i) = MU(i,1);
    end
end

d = R';
policy = pol;



update_S.m
-----------------------------------
function t_hat = update_S(pa, mu, i)
global S
global H
global P
mu_scal = vec_to_scalar(pa,mu);

if (H(i)+2) == P(i,1) && (rem(mu_scal(i),1) == 0)
    t_hat = S(:, :, i);
    for t=2:(pa.K - H(i))
        t_hat(H(i) + t, :) = P(i, 2:end);
    end
else
```

```
    t_hat = S(:, :, i);
    r = H(i) + 2;
    for c = 1:pa.N(i) + 1
        Q = exp(-pa.lambda(i)*pa.Delta*mu(i,r-1));
        P_rc = 0;

        for h = 0:c-1
            arr = c - 1 - h;
            P1 = (1-Q)^arr;
            P2 = Q^(pa.N(i)-arr);
            P3 = S(r-1,h+1,i);
            L = nchoosek(pa.N(i),arr);
            P_rc = P_rc + P1 * P2 * P3 * L;
        end

        t_hat(r,c) = P_rc;
    end
    for t = 3:(pa.K - H(i))
        t_hat(H(i) + t, :) = t_hat(H(i)+2,:);
    end

    P(i,1) = H(i) + 2;
    P(i,2:end) = t_hat(H(i)+2,:);
end



FD_greedy.m
---------------------------------
function r = FD_greedy(pa, S_hat)
F = 1;
```

```
for i = 1:pa.C % for every class i

    for r = 2:(pa.K)
        XSTAR = 0;

        for c = 1:(pa.N(i)+1)
            XSTAR = XSTAR+S_hat(r,c,i)*exp(-pa.lambda(i)*
            *pa.Delta*(c-1));
        end
        F = F * XSTAR;
    end
end

r = 1 - F;
```

```
checkbudget.m
-------------------------------
function b = checkbudget(pa,mu)
mu_scal = vec_to_scalar(pa,mu);
max_h = max(mu_scal);
spent_budget = 0;
for i=1:pa.C
    spent_budget = spent_budget+pa.N(i)*(1-exp(-pa.lambda(i)*
    *pa.Delta*sum(mu(i,:))))*pa.rho(i);
end

beta_vec = unique(pa.beta);
CB = 0;
```

```
for beac = 1:size(beta_vec,2)
Q = 0;
for j = 1:pa.C
if pa.beta(j) == pa.beta(beac)
cost = sum(mu(j,:));
if cost > Q
Q = cost;
end
end
end
CB = CB + Q * pa.beta(beac);
end

spent_budget = spent_budget + CB;
if spent_budget < pa.psi
    b = 1;
else
    b = 0;
end
```

## A.3 Agoritmo Greedy (versione 2)

greedy_cost_opt.m

```
-----------------------------------------
function [d policy] = greedy_cost_opt(pa,eps)
dt = pa.Delta;
kk = pa.K;
be_cost = pa.beta;
% variabili globali
global S
global H
global P

for e = 1:size(eps,2)
    pa.Delta = eps(e)*dt;
    pa.K = kk*(1/(eps(e)));
    pa.beta = eps(e)*be_cost;
    mu = zeros(pa.C,pa.K-1);
    H = zeros(pa.C,1);
    n_max = max(pa.N);
    P = NaN(pa.C, n_max + 2);
    S = NaN(pa.K, n_max+1, pa.C);

    for i=1:pa.C
        S(:,1:pa.N(i)+1,i) = 0;
        S(:,1,i) = 1;
        P(i,1:pa.N(i)+2) = [1 S(1,1:pa.N(i)+1,i)];
    end
    cc = 1;

    while (checkbudget(pa,mu)==1) && ~all(H==pa.K-1)&&
    && (cc >= 1)
```

```
mg = zeros(1,pa.C);
fix_S = S;
bf = FD_greedy(pa, fix_S);
bc = cost(pa,mu);

for i=1:pa.C

    if (H(i)+1) <= pa.K-1
        max_all = mu_hat_multi_tech(i,mu,pa);
        mu(i, H(i)+1) = min( [ 1  max_all ] );
        S_hat = S;
        T_hat = update_S(pa, mu, i);
        S_hat(:,:,i) = T_hat;
        af = FD_greedy(pa, S_hat);
        ac = cost(pa,mu);
        mg(i) = (af - bf)/(ac - bc);
        mu(i,H(i)+1) = 0;

    end
end

[maxval istar] = max(mg);

if (maxval == 0) && (H(istar) + 1 <= pa.K-1)
elseif (maxval == 0) && (H(istar+1) + 1 <= pa.K-1)
    istar = istar +1;
elseif (maxval == 0) && (H(istar+2) + 1 <= pa.K-1)
    istar = istar + 2;
end

if maxval>= 0
```

```
            cc = mu_hat_multi_tech(istar,mu,pa);
            mu(istar,H(istar)+1) = min( [ 1  cc ] );
            S_new = update_S(pa,mu,istar);
            S(:,:,istar) = S_new;
            H(istar) = H(istar)+1;
        end
    end

    S_final = S;
    R(e) = FD_greedy(pa, S_final);
    MU = vec_to_scalar(pa,mu);

    for i=1:pa.C
        pol(e,i) = MU(i,1);
    end
end

d = R';
policy = pol;



cost.m
------------------------
function c = cost(pa,mu)
cost = 0;
for i=1:pa.C % for each class
    cost = cost+pa.N(i)*(1-exp(-pa.lambda(i)*
    *pa.Delta*sum(mu(i,:))))*pa.rho(i);
end
c = cost;
```

## A.4 Algoritmo Greedy sulla frequenza di arrivo (cascata)

cascata.m

```
---------------------------------
function [d policy] = cascata(pa)
[lam ind] = sort(pa.lambda,'descend');
mu = zeros(pa.C,1);
h = hstar(pa);
mu_vec = scal_to_vec(pa,mu);

for i=1:size(lam,2)
    c = mu_hat_multi_tech(ind(i),mu_vec,pa);
    mu(ind(i),1) = min(c , pa.K-1);
    mu_vec = scal_to_vec(pa,mu);
    if c < pa.K-1
        break
    end
end

d = FD(pa,mu_vec);
policy = mu';
```

## A.5 Algoritmo delle politiche indipendenti

one_policy_multi_tech.m

```
-----------------------------------------------
function [b policy] = one_policy_multi_tech(pa)
h = sym('h');
T1 = 0;
T2 = 0;
T3 = 0;
T4 = pa.psi;

for i=1:pa.C
    T1 = T1+pa.N(i)*exp(-pa.lambda(i)*pa.Delta*
    *h)*pa.rho(i);
end

for i=1:pa.C
    T2 = T2 + pa.N(i) * pa.rho(i);
end

beta_vec = unique(pa.beta);
for b = 1:size(beta_vec,2)
    T3 = T3 + pa.beta(b);
end

g_h = T1 - T2 - T3 * h + T4;
[sol,nit]=newton(g_h,0,0.0001,100);

if sol > pa.K-1
    sol = pa.K-1;
end
```

```
for i=1:pa.C
    h_star(i,1) = sol;
end

p = scal_to_vec(pa,h_star);

if h_star>=0
    f = FD(pa,p);
else
    f = 0;
end

b = f;
policy = h_star;
```

## A.6 Algoritmo UB

ub_multiclass.m

```
-------------------------------------------
function [F_ub pol_ub]= ub_multiclass(pa,eps)
dt = pa.Delta;
kk = pa.K;
be_cost = pa.beta;

for e = size(eps,2)
    pa.Delta = eps(e)*dt;
    pa.K = kk*(1/(eps(e)));
    pa.beta = eps(e)*be_cost;

    global F
    global B
    global c_f
    global pol
    global ord

    pol = NaN(1,pa.C-1);
    F = 0;
    B = NaN(pa.C-1,pa.K);

    for f = 1:pa.C
        ord = 1:pa.C;
        c_f = f;
        ord(f) = [];
        path = NaN(1,pa.C-1);
        F_hat = VALUTA(pa, path);
    end
```

```
    for w = 1 : pa.C
        if (rem(pol(1,w),1)) > 0
            mu_ub_f(w,1) = ceil(pol(1,w));
        else
            mu_ub_f(w,1) = pol(1,w) + 1;
        end
    end

    l = scal_to_vec(pa,mu_ub_f);
    F_ub = FD(pa,l);
    pol_ub = vec_to_scalar(pa,l);
end
```