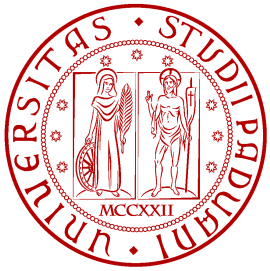

Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea Triennale in Ingegneria dell'Informazione

Instradamento egoistico in reti multi-salto

Laureando:
Giovanni Callegaro
Matricola 609404

Relatore:
Leonardo Badia

Anno Accademico 2011–2012

Sommario

Questa tesi discute il problema dell'instradamento egoistico nelle reti di comunicazione. Viene dapprima presentato lo stato dell'arte di tale problema illustrando cos'è stato studiato fino ad ora. Si introducono i concetti di base quali il costo totale, o sociale, gli equilibri di Nash e il prezzo dell'anarchia.

Successivamente viene descritta una metodologia di valutazione del prezzo dell'anarchia tramite il software MATLAB. Sono riportati e spiegati passo passo gli algoritmi necessari al calcolo del flusso ottenuto tramite instradamento egoistico e del flusso ottimale.

In seguito sono riportati i risultati di simulazioni svolte su diversi esempi significativi ricorrendo ad ampio uso di grafici che verranno discussi qualitativamente. Verrà infine discusso il vantaggio o meno della presenza di controllo centralizzato nelle reti.

Indice

1 Introduzione

Il problema dell'instradamento, o *routing*, nelle reti non riguarda solo le telecomunicazioni. Lo scopo è quello di individuare un percorso tra due nodi dati, uno di partenza ed uno di arrivo. Il percorso da generare può essere più o meno complesso ma in generale è una sequenza di nodi della rete che, se seguita ordinatamente, porta a buon fine la comunicazione. Si pensi al ruolo di un navigatore stradale, ciò che è richiesto di fare non è limitato all'elencare le direzioni da prendere ma di calcolare il percorso ottimo per arrivare a destinazione. C'è da chiedersi allora cosa si intenda per ottimo, infatti uno dei tanti percorsi da sorgente a destinazione non è in generale un buon risultato. Il percorso ottimo può essere definito da vincoli come il percorrere meno strada, l'impiegare meno tempo, evitare i pedaggi e simili. Per poter affrontare questo genere di richieste si è soliti definire funzioni di costo sui rami della rete, le quali possono dipendere da molti fattori diversi. È ora più semplice poter risolvere il problema del percorso ottimo: non è altro che il percorso la cui somma dei singoli elementi di strada risulta minore di tutti gli altri. Calcolare questo percorso è un problema già risolto nel campo dell'informatica: algoritmi come quelli di Bellman-Ford e Dijkstra permettono di ottenerlo nei migliori tempi possibili [1].

Ciò che verrà discusso in questa tesi riguarda il routing dinamico quando è presente commutazione di pacchetto. Routing dinamico significa che ogni nodo mantiene in memoria non solo la topologia della rete, ma anche il suo stato attuale, cioè informazioni sul traffico sui rami e sulla rimozione o aggiunta di collegamenti in modo da essere in grado di adeguare le proprie decisioni sullo stato della rete. Commutazione di pacchetto significa che la trasmissione è frazionata in piccole quantità chiamate pacchetti che possono essere instradati anche in percorsi diversi pur facendo parte di una stessa mole di dati. Nel caso che qualche pacchetto non arrivi a destinazione oppure che arrivino in ordine diverso da quello in cui sono stati divisi esistono diversi protocolli che si preoccupano di richiedere la trasmissione o di trovare loro il giusto ordine [2].

Il problema nasce da due possibili approcci all'instradamento dei pacchetti. Il primo suggerisce di cercare il percorso migliore per ognuno di questi e di instradarlo in tale modo; ogni pacchetto agisce quindi di testa propria non osservando il comportamento degli altri: questo è detto *instradamento egoistico*. La seconda strategia è quella di attuare un controllo centralizzato in maniera da indicare la percentuale di pacchetti da inviare in ogni possibile percorso. Questo caso è più complicato da gestire e richiede *signaling* aggiuntivo ma permette di ottenere sempre il costo totale minore possibile per la trasmissione. Questi due metodi infatti portano a risultati differenti e ci si

pone il problema di quale sia più opportuno utilizzare in generale, oppure, di individuare le situazioni più adatte per l'uno e per l'altro.

L'instradamento egoistico si rivela essere un caso particolare di giochi non collaborativi, studiati dalla *Teoria dei Giochi* [4]. Infatti verrà fatto ampio uso di termini provenienti da questa quali *equilibrio di Nash* e *prezzo dell'anarchia*. Quest'ultimo in particolare serve come metro di confronto tra l'approccio egoistico e quello collaborativo ed individua il fattore di perdita del costo totale utilizzando l'uno o l'altro metodo.

Nella seconda sezione viene introdotto il panorama in cui ci si muove parlando del problema e mostrando cos'è stato studiato a proposito fino ad ora. Vengono introdotti i concetti di base come il costo totale, o *sociale*, ed esposte le due reti chiave che fanno subito comprendere le fattezze che può assumere il problema: l'esempio di Pigou [5] ed il paradosso di Braess [6]. Pigou, economista inglese degli anni trenta, scoprì questo primo esempio nel 1920. Braess, matematico tedesco, portò alla luce il paradosso nel 1968. Questi problemi sono nati addirittura ben prima delle telecomunicazioni, da situazioni di tipo economico e di gestione stradale.

Nella terza sezione vengono illustrati e spiegati gli strumenti di studio utilizzati in questa tesi. Questi sono algoritmi scritti in MATLAB che forniscono con comodità i risultati da poter confrontare a proposito di applicare l'instradamento egoistico oppure agire mediante un controllo centralizzato. Vengono calcolati quindi percorsi, costi, e prezzo dell'anarchia di reti con funzioni di costo sui rami di tipo qualsiasi, per problemi di single-commodity. Il codice e lo pseudo-codice utilizzati sono commentati passo passo e vengono descritti ingressi e uscite degli algoritmi.

Nella quarta sezione sono riassunti e commentati i risultati più interessanti, ottenuti sempre per mezzo di MATLAB. È fatto ampio uso di grafici che mostrano le differenze tra le due modalità di gestione della trasmissione. Il parametro di confronto della differenza tra la bontà dell'uno e l'altro metodo in merito all'instradamento egoistico è, al solito, il prezzo dell'anarchia che viene più volte calcolato e discusso.

2 Stato dell'arte

Da quando esistono le reti esistono i problemi relativi al routing, cioè al trovare un percorso formato da altri elementi della rete che collaborano nel far arrivare i dati a destinazione. Le reti di calcolatori sono l'esempio più immediato ma non c'è da scordare che esistono numerose occasioni in cui questo problema si presenta, basti pensare ad una rete stradale o idrica. Emergono diversi problemi dalla natura delle reti: non è infatti trascurabile che alcuni collegamenti possano saltare all'improvviso oppure che un nodo possa congestionarsi. Inoltre i collegamenti presentano una capacità ben definita. C'è allora da dover gestire una situazione dinamica e non è più pensabile di poter risolvere problemi di questo genere a mano: ci si affida ai calcolatori che per monitorare lo stato istantaneo o pseudo-tale della rete fanno ampio utilizzo del signaling, ovvero traffico aggiuntivo che non aumenta l'entropia del segnale ma permette ai nodi della rete di sincronizzarsi e scambiarsi dati sullo stato di essa. Ma anche il signaling costa, inoltre è complicato da gestire. Si nota dunque che dietro ogni problema se ne nasconde un altro e tuttora si stanno studiando tecniche per risolvere la questione nel migliore dei modi.

2.1 Importanza del problema

Lo scopo delle tecniche di instradamento è di trovare il percorso in cui mandare un certo volume di traffico da un nodo di partenza ad uno di arrivo. Se i due nodi non sono collegati direttamente, o se lo sono ma esiste un secondo percorso più valido allora, in generale, si fa uso di algoritmi come quelli di Dijkstra o di Bellman-Ford per determinare il percorso migliore che il traffico dovrà seguire. *Migliore* però può avere molteplici significati, numericamente parlando. Il *prezzo* che presenta ogni strada può essere di natura diversa: un delay, un data loss, un percorso congestionato o perfino un costo monetario vero e proprio. Nell'applicare questi algoritmi il prezzo di ogni collegamento viene trattato come se fosse unico, di una stessa natura. Si assumi dunque che possa esistere un tasso di conversione da una tipologia di costo verso un'altra. Per risolvere il problema gli algoritmi di Dijkstra [8] e di Bellman-Ford [9] funzionano molto bene. Si dimostra che l'algoritmo di Dijkstra trova sempre il percorso migliore alla migliore complessità computazionale possibile a patto che i pesi sui rami siano non negativi [3]. Il problema però non si ferma qui: il traffico infatti non è un'unica unità ma si rappresenta in bit, frame, pacchetti etc. a seconda del livello a cui l'apparecchio trasmittente si trova a gestirlo. Nel nostro caso, il livello di rete, vengono chiamati pacchetti . In particolare però, usando un termine proprio della teoria dei giochi, possiamo pensare di suddividere il traffico in *agenti*. Il problema dell'instradamento

quindi non si ferma alla sola scelta del percorso migliore perché si dimostra che se ogni agente in cui è suddiviso il traffico compie la scelta di percorso suggerita da questi algoritmi, la migliore a suo modo di vedere, in generale non si ottiene il risultato migliore in termini complessivi. Questo tipo di routing è detto "egoistico" [11]. Al contrario, un approccio collaborativo tra gli agenti riesce a far minimizzare il costo totale. La differenza tra le due rendite pone una situazione tipica che sposa i problemi della teoria dei giochi.

2.1.1 Esempio di Pigou

La rete di Pigou è l'esempio classico del selfish routing.

Definiamo prima di tutto il termine *flusso* come l'insieme delle coppie sorgente-destinazione con associato, per ogni coppia, un vettore con le quantità di traffico che viaggiano per ogni diverso percorso possibile tra i due nodi.

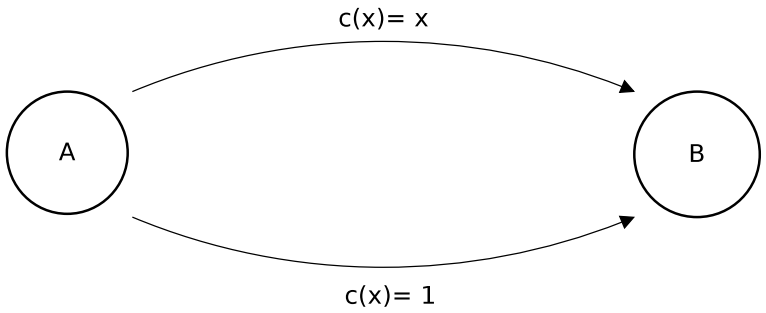

Figura 1: Grafo che rappresenta la rete associata all'esempio di Pigou

Il grafo di figura 1 rappresenta una rete formata dai due soli nodi di partenza e di arrivo A e B. Sono presenti due rami che collegano il nodo di partenza e quello di arrivo: quello più in basso ha costo di attraversamento 1 mentre quello in alto ha costo di attraversamento funzione del traffico che lo attraversa, in questo caso lineare. Il costo complessivo è la somma dei costi sui due rami pesata in base al traffico che li attraversa [4].

I $c(x)$ rappresentano i costi sui rami. Essi sono infatti funzione del traffico x che attraversa ognuno di essi.

Assumiamo per semplicità che la quantità di traffico che vogliamo trasmettere, a scanso di costanti di proporzionalità, valga 1.

Possiamo quindi definire, per ogni possibile flusso f da un nodo di partenza ad uno di arrivo, il costo di un percorso P:

$$c_P(x_P) = \sum_{e \in P} c_e(x_e) \cdot x_P$$

con x_e la quantità di traffico che attraversa ogni ramo e del percorso P secondo quanto dettato dal flusso f (infatti dipende anche dalle quantità di traffico che scorrono negli altri percorsi) e x_P la percentuale di traffico totale che sceglie di intraprendere il percorso P.

Il costo totale si ottiene facendo la sommatoria del costo di ogni percorso:

$$C(f) = \sum_P c_P$$

Nel caso dell'approccio egoistico ogni agente, quantità piccola a piacere di traffico, agisce per conto suo: applicando l'algoritmo di Dijkstra ogni agente sceglie il percorso con il ramo $c(x) = x$ perché in ogni valutazione si verifica che il costo per tale ramo è sempre minore di quello dell'altro. Infatti, per qualunque agente stessimo valutando, la quantità di traffico già instradata risulta minore di 1. La scelta viene ripetuta per ogni agente fino ad ottenere la situazione in cui tutti quanti hanno scelto tale ramo. Allora il flusso prevede che nel percorso in alto l'affluenza sia il 100% e nell'altro ramo sia nulla. A questo punto il costo totale risulta essere:

$$C(f) = (c(1) = 1) \cdot 1 + 1 \cdot 0 = 1$$

Se, al contrario, si scegliesse di inviare quantità di agenti diverse sui due percorsi, ad esempio dividendo il traffico in maniera uguale, il risultato cambierebbe. In questo caso il costo totale risulta:

$$C(f) = \left(c\left(\frac{1}{2}\right) = \frac{1}{2}\right) \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} = \frac{3}{4} = 0.75$$

Il costo totale è stato ridotto quindi del 25%. Si dimostra inoltre che, in questo esempio, 0.75 è il risultato migliore che si possa ottenere.

Introduciamo ora due concetti fondamentali: equilibrio di Nash e prezzo dell'anarchia.

L'equilibrio di Nash [10] è un profilo di strategie (una per ciascun giocatore) rispetto al quale nessun giocatore ha interesse ad essere l'unico a cambiare. Senza incorrere in definizioni proprie della teoria dei giochi limitiamoci al nostro caso del traffico sull'esempio di Pigou: quando ogni agente del traffico ha deciso di instradarsi nel primo percorso, per ognuno di questi non ha più senso pensare di scegliere un'altra strada perché il costo sul percorso alternativo di certo non diminuirebbe. Questo porta l'intero sistema a trovarsi in uno stato di equilibrio in cui ogni agente(giocatore) ritiene più opportuno non cambiare strategia(percorso). Nel caso di Pigou, di equilibri di Nash ce n'è uno solo ma in realtà si possono trovare facilmente esempi in cui questi sono più di uno.

La definizione di Prezzo dell'Anarchia [7] secondo la teoria dei giochi è la seguente: Dati un gioco G e una funzione sociale f (somma delle funzioni di payoff di tutti i giocatori), sia N l'insieme di tutti gli equilibri di Nash e sia OPT lo stato di G che ottimizza f. Il prezzo dell'anarchia del gioco G rispetto ad f è definito come:

$$PoA_G(f) = \sup_{s \in N} \frac{f(s)}{f(OPT)}$$

Esso misura la perdita di ottimalità di un sistema non regolato a causa della mancanza di cooperazione tra i giocatori e di coordinazione centrale. Spesso viene anche chiamato *worst-case coordination ratio*.

In altre parole nel nostro caso per trovare il prezzo dell'anarchia bisogna scovare il peggiore tra gli equilibri, peggiore nel senso di costo totale che comporta, e farne il rapporto con il costo del flusso ottimale.

Otteniamo allora

$$PoA = \frac{1}{0.75} \simeq 1.3333$$

2.1.2 Paradosso di Braess

Riguardo all'instradamento egoistico una situazione molto interessante da far notare è il paradosso di Braess, che può avere effetti davvero indesiderati al lato pratico: Si presume di voler trasmettere da un nodo A ad un nodo D una certa quantità di traffico che, a scanso di costanti di proporzionalità, assumiamo in quantità totale 1.

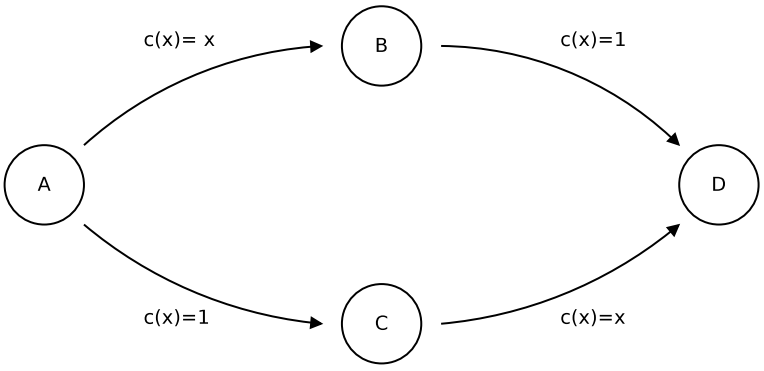

Figura 2: Situazione iniziale del paradosso di Braess: i carichi vengono suddivisi simmetricamente

Secondo l'approccio egoistico ogni agente sceglie la strada meno percorsa visto che entrambe sono perfettamente identiche fino a che il traffico totale

si divide in due parti esatte. Il costo totale risulta quindi:

$$C(f) = \left(\frac{1}{2} + 1\right)\frac{1}{2} + \left(1 + \frac{1}{2}\right)\frac{1}{2} = \frac{3}{2} = 1.5$$

Ora proviamo a pensare di migliorarlo inserendo un ramo ideale di costo 0 nella rete tra i nodi B e C.

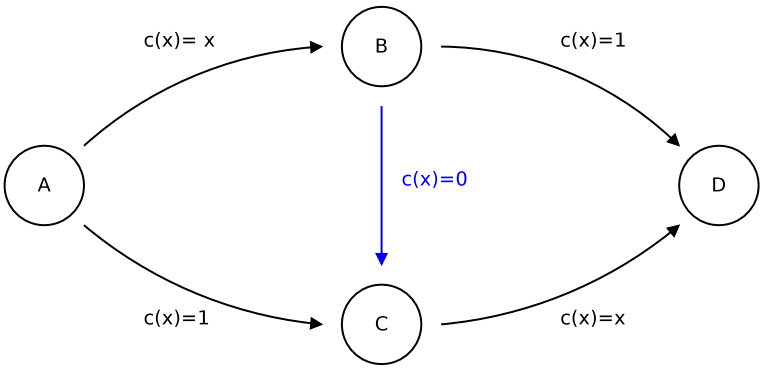

Figura 3: Viene aggiunto un collegamento a costo zero

A questo punto ogni agente, in ogni momento, sceglierà il percorso A-B-C-D in quanto, per ogni $0 < x < 1$,

$$x + 0 + x < x + 1$$

Quindi alla fine il costo totale risulta essere:

$$C(f) = c(1) + 0 + c(1) = 2$$

Non solo non migliora ma è **addirittura peggiorato** significativamente. Sembra essere un paradosso perché avendo inserito una strada aggiuntiva di costo 0 ci si aspetterebbe che il costo totale diminuisca. Invece aumenta addirittura, con il rischio di aver speso risorse per nulla.

2.1.3 Equivalente meccanico dell'instradamento egoistico

È interessante notare come questi problemi non siano limitati solo al campo delle telecomunicazioni o dove siano presenti reti in quanto tali. Viene presentato ora un analogo meccanico del paradosso di Braess mediante fili e molle [4].

La parte a) della figura rappresenta la seconda parte di grafo del paradosso di Braess. Pensando al costo totale come alla distanza tra il sostegno

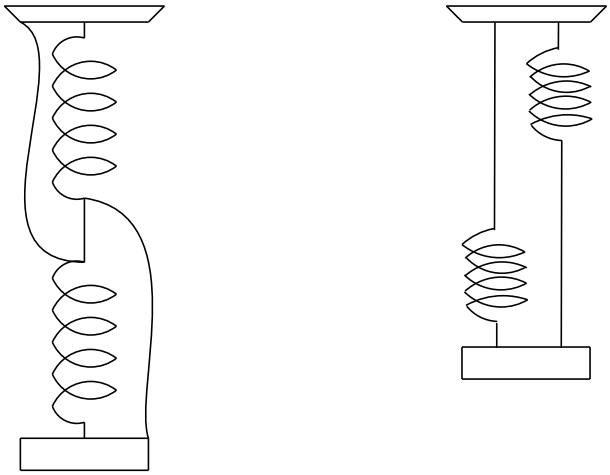

Figura 4: Analogo meccanico del paradosso di Braess mediante pesi, fili e molle

ed il peso, le molle corrispondono ai rami di costo $c(x) = x$ perché il loro allungamento è proporzionale alla tensione applicata ad esse mentre il rimanente del filo è di costo fisso perché indeformabile. Nella situazione b) il filo centrale viene tagliato: la tensione esercitata dalla forza di gravità si divide equamente nei due fili rimasti, quindi le molle sono sottoposte ad un allungamento minore. Si ottiene che il costo finale è quindi minore in questo secondo caso.

2.2 Related Works

Nelle *reti multi-salto* avviene che quando un nodo A deve trasmettere ad un nodo B, esso calcola solo il primo nodo del percorso, trasmette a questo, e lascia che siano gli altri nodi della rete ad occuparsi di tutto il resto. In altre parole ogni nodo è intelligente, nel senso che può calcolare anch'esso il resto del percorso rimanente.

Esistono allora diverse strategie per l'instradamento da poter applicare come il Flooding, il Distance Vector e il Link state.

Flooding [3] - ogni pacchetto di dati da trasmettere viene spedito ad ogni proprio vicino. Questi, se non sono i destinatari, si occupano di ritrasmetterlo ad ogni loro vicino ad eccezione di quello da cui proveniva l'originale. La tecnica funziona ma è disastrosa perché utilizza con scarsissima efficenza la rete. Per garantire che i pacchetti non vengano ritrasmessi all'infinito viene

inserito loro un contatore (TTL, o Time-To-Live) che descresce ad ogni passo e il cui valore iniziale è tale da permettere di poter raggiungere il nodo di destinazione. Quando il contatore raggiunge lo zero esso smette di essere ritrasmesso. Questa tecnica è semplice ma assolutamente non performante, tuttavia permette di raggiungere la destinazione nel minor tempo possibile. Viene utilizzata ad esempio per scopi militari, quando i nodi possono essere danneggiati con facilità, oppure come parte di altre tecniche che hanno bisogno di disseminare pacchetti per la rete, ad esempio per inizializzarla.

Distance Vector [12] - ogni nodo possiede una tabella che contiene, per ogni altro nodo della rete, il primo passo del percorso e la distanza. Questa tabella dev'essere costantemente aggiornata scambiando le informazioni dai propri vicini. Questa tecnica realizza una versione distribuita dell'algoritmo di Bellman-Ford. La condizione per realizzare il distance vector è che ogni nodo conosca la distanza dei propri vicini. Sorgono problemi quando i costi dei rami cambiano o alcuni collegamenti saltano: infatti le versioni delle *routing table* che circolano possono fare riferimento a dati vecchi e inviare informazioni errate quali l'esistenza di un certo percorso o meno e possono venire a crearsi quindi situazioni di loop conosciute come *count-to-infinity*.

Link State [13] - dopo aver calcolato le distanze coi propri vicini tramite pacchetti ECHO, ogni nodo le condivide con tutti gli altri tramite flooding. Vengono poi calcolati tramite l'algoritmo di Dijkstra i percorsi migliori. Se la topologia della rete cambia, ad esempio un collegamento salta, ci vuole comunque un po' di tempo perché Dijkstra ritorni una soluzione accettabile in quanto c'è da aspettare che il flooding avvisi che ci sono stati dei cambiamenti. Gli svantaggi del Link State sono che deve tenere in memoria l'intera mappa della rete e che ogni router deve avere a disposizione una discreta capacità di calcolo per richiamare Dijkstra ogni volta. Inoltre, per il flooding che è comunque sempre presente e il fatto che i dati da scambiarsi e tenere in memoria sono molti, gestire le reti di grandi dimensioni è faticoso. È per queste che è previsto lo *Hierarchical Routing* [14] per il quale la rete viene divisa in sottoreti che si possono pensare ancora come a nodi di una rete normale. Ogni sottorete poi può utilizzare tecniche a sé stanti anche diverse tra loro.

Riguardo il problema del **selfish routing** sono stati scritti numerosi articoli. In questi viene fatto ampiamente uso di analogie coi problemi della teoria dei giochi. Il selfish routing infatti non è altro che un particolare gioco *non atomico* non cooperativo. L'insieme di giocatori (gli agenti) non è finito ed ogni giocatore ha contributo infinitesimale.

In [17, 19] viene discussa l'esistenza degli equilibri di Nash e costruiti algoritmi per calcolarli ed individuare quindi quelli di impatto migliore e peggiore. In [23] tramite un'analisi matematica viene individuato con precisione il prezzo dell'anarchia in reti abbastanza semplici con funzioni di costo lineari, sapendo la differenza di costo dal ramo più veloce a quello più lento e il numero di rami m.

Non per tutte le funzioni di costo si dimostra essere possibile trovare un limite numerico al prezzo dell'anarchia: un controesempio [4] lo si trova subito pensando alla rete dell'esempio di Pigou con una funzione di costo $c(x) = 1$ e l'altra non lineare $c(x) = x^p$ per p molto grande. Il costo totale che si ottiene applicando la strategia ottima, collaborativa, risulta $C(f) = 1-p\cdot(p+1)^{-\frac{p+1}{p}}$. All'aumentare di p questo può diventare arbitrariamente piccolo e quindi il prezzo dell'anarchia arbitrariamente grande. Infatti $\lim_{p\to\infty} C(f) = 0$.

È possibile discutere l'efficacia dell'applicare o meno tecniche centralizzate per la gestione del traffico. In [21, 22] si dimostra che, se consideriamo reti in cui il traffico cambia dinamicamente, per la quasi totalità di esse il danno prodotto dal selfish routing varia in maniera *logaritmica* con un fattore di sensitività della variazione del traffico. Vengono studiati casi di reale comportamento nelle reti con i rami della rete aventi costi per il delay di tipo M/M/1 e i risultati sono pressoché invariati. Viene fatto notare invece che algoritmi per il controllo delle congestioni svolti dai nodi di partenza, come quelli che prevede TCP, riescono a ridurre di molto le perdite dovute all'instradamento egoistico. Studiando reti in cui l'instradamento non è egoistico con un approccio analitico e non proprio della teoria dei giochi si scopre che in certi casi i flussi suggeriti sono addirittura gli stessi di quelli che gli agenti prenderebbero autonomamente. Al contrario di quanto si può pensare dalla teoria, che guarda spesso ai casi peggiori, mediamente l'instradamento egoistico si avvicina di molto ai risultati dei flussi ottimali. Comunque avere risultati di questo tipo porta spesso ad una forte congestione su alcuni rami e inoltre la natura *adattativa* degli agenti rende il traffico sulla rete meno prevedibile [16].

È possibile notare dal paradosso di Braess che in certi casi rimuovere un collegamento alla rete riesce a migliorare il costo totale quando l'instradamento è egoistico. È quindi un'idea provare a rimuovere rami dalla rete in maniera intelligente per ottenere una sottorete dalle migliori prestazioni.

In particolare viene introdotto il problema della *nashification* [15] cioè, dato un flusso generico, modificare la rete in modo che questo venga portato ad essere all'equilibrio di Nash. In questo modo, creando un flusso all'equili-

brio di Nash partendo da un flusso ottimale è possibile preservare il vantaggio del basso costo totale che si otterrebbe cooperativamente. Questi algoritmi che si occupano di cercare migliori design per la struttura della rete trovano complessità computazionale di $O(n \cdot m^2)$ con n il numero di utenti che devono trasmettere ed m il numero di percorsi relativi. Algoritmi di questo tipo sono applicabili sia al caso in cui il traffico da sorgente a destinazione si può dividere in più percorsi, sia quelli per cui ne si deve scegliere uno solo.

Un modo possibile per modificare la rete dinamicamente è quello di *tassare* o, meglio, penalizzare alcuni percorsi per influenzare i risultati degli algoritmi di instradamento degli agenti [4] [18].

Queste penalizzazioni ovviamente devono ritornare, direttamente o indirettamente, a beneficio degli utenti per non perdere efficenza. Vengono ideate quindi strategie per penalizzare i rami della rete al fine di, quando ci si trova in una situazione di equilibrio di Nash, ridurre i costi al minimo. Ci si basa sul metodo delle *funzioni di costo marginali* [4].

Se c è una funzione di costo differenziabile allora la corrispondente funzione di costo marginale c' è definita da:

$$c' = \frac{d}{dx}(x \cdot c(x))$$

Si dimostra che su una rete con funzioni di costo continue, differenziabili e semiconvesse allora un flusso f è ottimale per la rete se esso è un flusso all'equilibrio di Nash per la stessa rete ma a cui sono stati sostituite le funzioni di costo con le relative marginali [4].

Si riesce ad ottenere un effetto simile a quello descritto prima ma più potente perché non viene previsto unicamente il rimuovere rami ma anche di scoraggiarne l'utilizzo in maniera controllata [25].

Ad esempio, una tassa adeguata sul ramo a costo zero del grafo del paradosso di Braess permette di eliminarlo virtualmente ma se si volesse si potrebbe anche semplicemente limitarne l'utilizzo. Tramite questo sistema di penalizzazione dei percorsi, si è riusciti a migliorare l'efficenza per una vasta gamma di reti, in particolare per tutte le reti con costi sui rami lineari. Viene anche notato che in queste ultime il guadagno dall'introdurre penalizzazioni non supera mai quello rispetto al rimuovere rami mentre in molte reti con funzioni di costo non lineari il sistema di tassazione è addirittura molto più potente.

Esistono strategie ibride che gestiscono il caso in cui sulla rete siano presenti sia elementi egoistici sia un controllo centralizzato. Queste sono studiate basandosi sugli *Stackelberg Games* della teoria dei giochi [4].

In un interessante articolo si è pensato ad un algoritmo per identificare i nodi problematici, non collaborativi, tramite analisi statistiche in reti con protocollo di routing AODV(Ad-hoc On-demand Distance Vector) ottendendo ottimi risultati ed un basso numero di falsi positivi [24].

In [26] si sono studiati i tempi di convergenza verso gli equilibri di Nash. In particolare risulta che per raggiungere un equilibrio ϵ-approssimato il tempo che ci vuole è polinomiale per ϵ. I risultati vengono anche estesi ai problemi di multi-commodity (*asymmetric* games).

Basandosi su un metodo di studio di giochi non cooperativi con informazioni incomplete [27] è stato pensato di definire il gioco dell'instradamento egoistico con informazioni incomplete. Questo prende il nome di *Bayesian routing game* [20]. Ci sono dunque n nodi devono instradare il loro traffico attraverso m percorsi. Ogni nodo agisce secondo un proprio carattere e non conosce le quantità di traffico che gli altri trasmettono, quindi deve fare riferimento ad un modello probabilistico descritto da una certa funzione di distribuzione che individua le diverse possibilità tra i caratteri degli altri nodi. Il modello pone ottimi risultati: si riescono ad individuare comunque equilibri di Nash e in alcuni casi anche a calcolarli in tempo polinomiale oltre a trovare i limiti per il *coordination ratio*.

3 Descrizione contributo

In questa sezione si presentano degli algoritmi di calcolo per costi e prezzo dell'anarchia in reti multi-salto che possono essere scelte arbitrariamente. L'operato è diviso in tre punti.

Per prima cosa viene compiuta una simulazione tramite MATLAB dell'instradamento egoistico.

In secondo luogo si usa il toolbox di ottimizzazione vincolata per cercare il flusso ottimale.

Successivamente viene calcolato il prezzo dell'anarchia.

Tutti i dati raccolti in questi passaggi forniranno la base delle osservazioni esposte nella sezione dei risultati.

L'unica ipotesi che vige è che il codice sviluppato non prevede di gestire i problemi di multi-commodity.

3.1 Dati in ingresso e descrizione della variabili

I dati di ingresso per l'algoritmo sono due: il grafo della rete, il nodo di partenza e il nodo di arrivo del traffico.

Grafo della rete: per poter rappresentare un grafo in matlab viene fatto uso di una matrice quadrata. La righe della matrice sono tante quante i nodi del grafo. Nella cella (i, j) della matrice viene rappresentato il ramo direzionato che parte dal nodo i e arriva al nodo j. Più precisamente la cella (i, j) contiene un dato di tipo *cell* che rappresenta una stringa di caratteri. In questo modo per inserire una funzione di costo su un ramo è necessario solamente scrivere per esteso una funzione del traffico $c(x)$. Questo permette di poter cambiare con facilità le funzioni di costo ma soprattutto di averle arbitrarie. Non si pone alcun limite quindi alle tipologie di funzioni da poter inserire.

Si noti che:

- Il costo -1 sta ad indicare che non c'è collegamento. Ciò va bene perché -1 è un costo fisicamente impossile da avere per un ramo. Inoltre è fuori dal dominio per l'algoritmo di Dijkstra che non prevede rami con costi negativi.

- Sorge il problema che sulla rete possano esserci due o più rami che vanno dal nodo A al nodo B. Per aggirare il problema è sufficiente introdurre un nodo fittizio C, per ogni ramo doppio, non collegato a nessun altro nodo della rete esclusi A e B. Il collegamento tra A e C deve avere costo come previsto dal ramo che si sta sostituendo mentre il ramo da C a B deve avere costo 0. Oppure il viceversa.

Per fare chiarezza verranno riportati il codice e la matrice relativi al grafo dell'esempio di Pigou, in figura 5.

```
grafo=[cellstr('0'), cellstr('x'), cellstr('1');
    cellstr('-1'), cellstr('0'), cellstr('-1');
    cellstr('-1'), cellstr('0'), cellstr('0')];
```

Matrice corrispondente:

$$\begin{bmatrix} 0 & x & 1 \\ -1 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix}$$

Grafo associato:

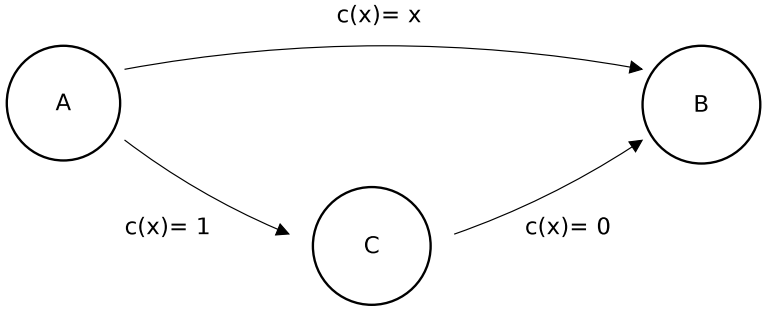

Figura 5: Grafo dell'esempio di Pigou modificato per evitare rami multipli tra due nodi

Nodo di partenza: visto che il nodo di partenza, secondo le ipotesi, è uno solo viene utilizzato di default il nodo con indice 1.

Nodo di arrivo: l'indice del nodo di arrivo deve essere specificato all'inizio del codice.

3.2 Instradamento egoistico

Per simulare ciò che avviene in una rete quando vige l'approccio egoistico bisogna far scegliere ad ogni agente la propria strada. Il comportamento dell'agente $n+1$ si baserà non solo sulla topologia della rete ma anche sulle sue condizioni determinate dalle scelte dei primi n agenti.

Il programma prevede di far eseguire l'algoritmo di Dijkstra per ogni nuovo agente che si pone il quesito di che percorso compiere. Questa porta sicuramente ad una situazione di equilibrio perché, in ogni momento, se un agente già instradato potesse tornare indietro a chiedersi qual è il percorso

migliore da poter compiere si ritroverebbe a fare la stessa scelta presa in precedenza. In altre parole, quella a cui si porteranno gli agenti può essere considerata un'allocazione egoistica. Il fatto che il costo sui rami nei reali sistemi fisici per i quali si esegue questa procedura possa non essere mai descrescente porta a poter ritenere l'equilibrio trovato quello di costo minore tra i diversi equilibri possibili [4].

La funzione *dijkstra()* riceve la matrice *pesiSuiRami* che contiene i valori numerici dei costi attuali. Il nodo 1 è il nodo iniziale da cui calcolare i percorsi minimi verso ogni altro nodo. Ritorna il vettore *preced* che contiene, per ogni elemento tranne il primo (0 per default), il nodo precedente nel cammino minimo. Se il nodo precedente non esiste (perché il grafo non è connesso) inserisce -1.

---

function [preced] = dijkstra (pesiSuiRami)
crea nuvola= vettore che $\forall$ nodo indica la sua presenza nella nuvola;
inserisci il nodo iniziale nella nuvola;
crea preced;
crea distanze= vettore con le distanze dal nodo iniziale;
for all i nodo attiguo a quello iniziale **do**
    $distanze(i) \leftarrow pesiSuiRami(1, i)$;
    $preced(i) \leftarrow 1$;
end for
while nuvola non piena **do**
    if $\forall j \notin$ nuvola, j non raggiungibile **then**
        termina;
    end if
    cerca min, nodo esterno di distanza minore;
    inserisci min nella nuvola;
    for all $i \notin$ nuvola, i attiguo a min **do**
        if i non raggiungibile $\vee$ $distanza(i) \geq pesiSuiRami(min, i)$+$distanze(min)$
then
            $distanza(i) \leftarrow pesiSuiRami(min, i) + distanze(min)$;
            $preced(i) \leftarrow min$;
        end if
    end for
end while
return preced
end function

All'inizio della simulazione devono venire calcolati tutti i percorsi possibili dal nodo di partenza a quello di arrivo e memorizzati in un matrice opportuna. L'algoritmo che trova i percorsi *trovapercorsi()*, che ha bisogno di richiamare una funzione ricorsiva *trovapercorsiInterno()*, ritorna *percorsi*, matrice con numero di colonne uguale alla dimensione del grafo iniziale (lunghezza massima dei percorsi), che contiene sulle righe tutti i percorsi possibili dal nodo 1 al nodo finale. Quando i percorsi terminano prima di aver esaurito la riga il rimanente dello spazio, per semplicità, viene riempito di -1.

```
function [percorsi] = trovapercorsi(grafo, finale)
crea percorsi come variabile globale;
percorsi ← matrice vuota;
strada ← [1];
trovapercorsiInterno(grafo, finale, strada);
return percorsi
end function

function trovapercorsiInterno(grafo, finale, strada)
for all i ∈ grafo do
    if i ha un collegamento con l'ultimo nodo di strada then
        if i = finale then
            aggiungi i a strada;
            aggiungi strada come nuova riga di percorsi;
            rimuovi l'ultimo elemento da strada;
            continua con il prossimo step del ciclo;
        end if
        if i ∉ strada then
            aggiungi i a strada;
            esegui trovapercorsiInterno(grafo, finale, strada);
            rimuovi l'ultimo elemento da strada;
        end if
    end if
end for
end function
```

Nel codice, *strada* è un vettore di dimensione variabile che svolge il ruolo di ricordare la sequenza di nodi già attraversata. L'algoritmo si preoccupa di rimuovere e aggiungere elementi da *strada* in modo da ottenere tutte le combinazioni. I possibili cicli vengono evitati controllando se i nodi da aggiungere non siano già presenti. Tutti i percorsi senza cicli che portano dal nodo ini-

ziale a quello finale che si presentano in strada vengono dunque memorizzati in una nuova riga di *percorsi*.

Definite queste due funzioni, inizia la parte principale del codice: l'unità di traffico viene suddivisa in un numero molto alto di agenti per potersi avvicinare di molto all'effetto di trascurabilità del peso di ciascuno.

Viene creato inoltre un vettore, all'inizio nullo, che per ogni diverso percorso memorizza la quantità di traffico che viene inviata a mano a mano.

Dopo una fase di inizializzazione, per ogni agente di traffico che scruta la rete viene fatto eseguire Dijkstra che individua il percorso minimo in ogni momento e vengono aggiornati questi tre elementi sulla base della sua scelta e di quelle degli agenti precedenti:

- La quantità di traffico di ogni percorso in *traffici*.
- La quantità di traffico per ogni ramo in *trafficoSuiRami*.
- Il costo di ogni ramo, funzione della sua quantità di traffico, in *pesiSuiRami*.

```
for i=1:1:nAgenti
   preced= dijkstra(pesiSuiRami);
   percorsoScelto= cercaPercorso(preced, arrivo, percorsi);
   traffici(percorsoScelto)= traffici(percorsoScelto) + agente;
   for j=2:1:dim(2)
      if percorsi(percorsoScelto, j)==-1
         break;
      end
      trafficoSuiRami(percorsi(percorsoScelto, j-1),
        percorsi(percorsoScelto, j)) =
        trafficoSuiRami(percorsi(percorsoScelto, j-1),
        percorsi(percorsoScelto, j)) + agente;
      x= trafficoSuiRami(percorsi(percorsoScelto, j-1),
        percorsi(percorsoScelto, j));
      pesiSuiRami(percorsi(percorsoScelto, j-1),
        percorsi(percorsoScelto, j))=
        eval(grafo{percorsi(percorsoScelto, j-1),
        percorsi(percorsoScelto, j)});
   end
end
```

cercaPercorso() è una semplice funzione che ritorna l'indice della riga della matrice *percorsi* corrispondente al tragitto che l'algoritmo di Dijkstra suggerisce.

Dopo aver eseguito questo algoritmo per tutti gli agenti così che l'intera unità di traffico si è disposta nei percorsi localmente migliori, disponiamo di tutte le informazioni che servono per calcolare il costo totale mediante una sommatoria sui rami, del prodotto tra il loro costo e il traffico che li attraversa.

---

costoSoluzioneSelfish $\leftarrow 0$;
for all $i \in$ grafo **do**
  for all $j \in$ grafo **do**
    costoSoluzioneSelfish = costoSoluzioneSelfish +
    trafficoSuiRami(i, j) $\times$ pesiSuiRami(i, j);
  end for
end for

3.3 Ricerca del flusso ottimale

Per cercare il flusso ottimale viene utilizzato l'optimization toolbox già presente in MATLAB.

Dobbiamo quindi individuare la funzione da minimizzare, i vincoli, il vettore degli ingressi, un punto di partenza e le altre opzioni necessarie.

La funzione da minimizzare è quella che calcola il costo totale. La sintassi del toolbox le permette di avere in ingresso solo un vettore, quello dei traffici dei diversi percorsi. Gli altri dati necessari vengono passati utilizzando variabili global.

La funzione viene realizzata in tre passi:

- Viene calcolato il traffico su ogni ramo a partire da quello sui percorsi scansionando la matrice di questi ultimi e sommando di volta in volta.
- Per ogni ramo ne viene calcolato il costo, valutando la funzione $c(x)$ in base al proprio traffico.
- In maniera analoga a quanto visto per l'instradamento egoistico viene svolta una sommatoria dei costi di ogni ramo pesati per il traffico che li attraversa.

---

function costoCollab = daMinim(traffico)

recupera le variabili global;
for all $i \in$ percorsi **do**
  for all j ramo del percorso i **do**
    trafficoSuiRami(j) = trafficoSuiRami(j) + traffico(i);
  end for
end for
for all i riga del grafo **do**
  for all j colonna del grafo **do**
    x= trafficoSuiRami(i, j);
    pesiSuiRami(i, j)= la propria funzione valutata in x;
  end for
end for
costoCollab $\leftarrow$ 0;
for all $i \in$ percorsi **do**
  for all j ramo del percorso i **do**
    costoCollab = costoCollab + pesiSuiRami(j) $\times$ traffico(i);
  end for
end for
return costoCollab
end function

I vincoli per il toolbox si dividono in due categorie: lineari e non lineari. Per i lineari si devono specificare una matrice A e un vettore b tali che $Ax <= b$. Il seguente codice impone che i traffici per ogni percorso siano non negativi.

```
% Matrici per i vincoli lineari di disuguaglianza Ax <= b
A= zeros(numeroPercorsi, numeroPercorsi);
for i=1:1:numeroPercorsi
    A(i, i)= -1;
end
b= zeros(numeroPercorsi, 1);
```

I vincoli non lineari vanno specificati in una funzione a sé.

Per i non lineari di disugualianza non ci sono vincoli. Per i non lineari di ugualianza l'unico vincolo da dover rispettare è che la norma dei traffici deve essere 1.

```
function [c, ceq]= nonlinconstr(x)
    % Vincoli di disuguaglianza non lineari (visti come c <=0)
    c= []
    % Vincoli di uguaglianza non lineari (visti come ceq ==0)
    ceq = sum(x) -1;
```

```
    return
end
```

Dopo aver calcolato i percorsi con la stessa funzione vista in precedenza per l'instradamento egoistico, viene creato il vettore iniziale da cui il toolbox deve partire. Per semplicità viene preso il caso in cui il primo percorso è a pieno regime e gli altri non vengano mai utilizzati.

```
x0= [1; zeros(numeroPercorsi-1, 1)];
```

Le opzioni che il toolbox richiede permettono di modificare la precisione, il numero di tentativi, le modalità per fermarsi, velocizzare la ricerca se si conosce già il gradiente della funzione da minimizzare e molte altre possibilità. In generale il seguente set funziona in maniera sufficentemente precisa.

```
options = optimset ('LargeScale', 'off', 'MaxFunEvals', 1000,
     'GradObj', 'off', 'TolFun', 1e-9 , 'TolX', 1e-9 ,
     'TolCon', 1e-6, 'Display ', 'iter');
```

La minimizzazione vera e propria viene svolta dall'unica riga di codice che segue.

```
[xopt, fval, exitFlag] = fmincon(@(x)daMinim(x), x0 , A , b,
     [], [], [], [], @(x)nonlinconstr(x), options);
```

I valori restituiti dal toolbox sono tre: *xopt* contiene il vettore dei traffici che permette di avere il costo totale più basso, contenuto in *fval*. La variabile *exitflag* invece contiene informazioni sullo svolgimento andato a buon fine o meno della ricerca.

3.4 Calcolo del prezzo dell'anarchia

Il *prezzo dell'anarchia* è calcolato molto semplicemente tramite una banale divisione.

```
PoA= costoSoluzioneSelfish/fval;
```

4 Risultati

Vengono illustrati ora diversi risultati dell'analisi tramite MATLAB.

Per tutte le simulazioni è stata utilizzata la versione R2011a per Unix su una macchina dotata di un core i7 a 1.60 GHz.

Inizialmente verrà mostrato il comportamento di alcuni esempi basati su Pigou e la sua versione non lineare. Poi simuliamo cosa succede quando vengono instradati agenti di dimensione non più trascurabile. Viene in seguito ripreso il caso del paradosso di Braess, il caso di una rete con numerosi nodi e rami e il caso in cui siano presenti rami di costo esponenziale, situazione che si avvicina molto più alla realtà rispetto alle altre.

Prendiamo allora l'esempio di Pigou: facciamo variare l'esponente p della funzione di costo $c(x) = x^p$ e mostriamo come varia il costo totale in base al traffico sui soli due percorsi possibili.

Tracciamo il grafico del costo per $p = 1$ e $p = 4$ al variare del traffico in figura 6, secondo il flusso $[x, 1 - x]$ con $0 \leq x \leq 1$.

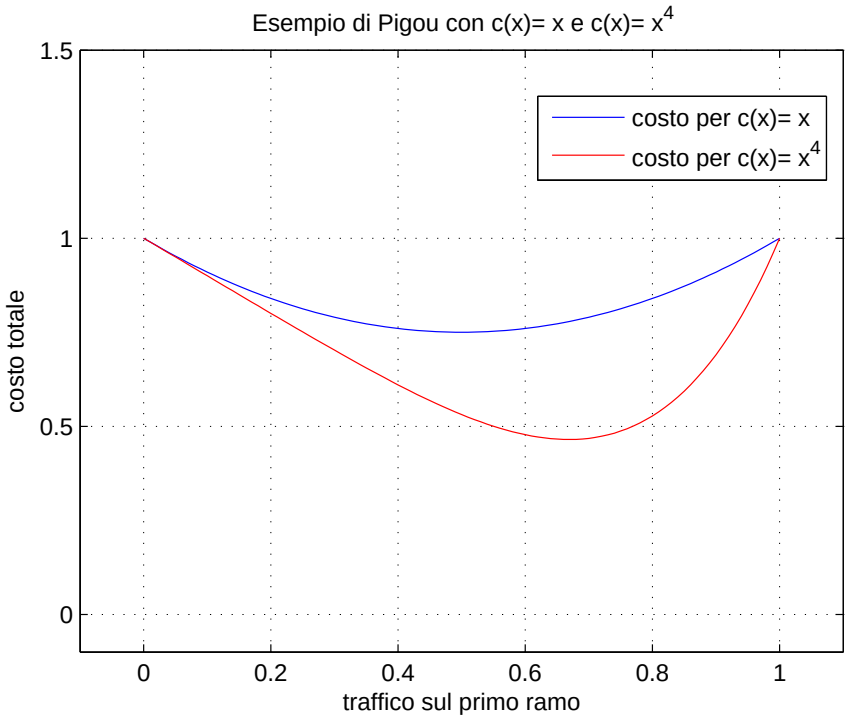

Figura 6: Variazione del costo in base al traffico sui due percorsi.

Sulle ascisse è presente il traffico sul primo percorso (quello con $c(x) = x$ o $c(x) = x^4$). Quello sul secondo è semplicemente il corrispettivo complemento a 1. Sulle ordinate c'é il costo totale per il corrispettivo flusso, nei due casi.

La simulazione per $p = 1$ restituisce i seguenti dati:

Percorsi	Traffico	Traffico ottimale
1-2	100%	50%
1-3-2	0%	50%

Costo totale	1
Costo ottimale	0.75
Prezzo dell'anarchia	1.3333

Osserviamo che $[0.5, 0.5]$ è effettivamente il punto di minimo.

Il grafico ottenuto è una parabola, il costo quindi non si sposta linearmente rispetto alla differenza di quantità di traffico nei percorsi ma secondo il quadrato di esse. Che sia una parabola può essere velocemente verificato dai conti:

$$C(x) = x \cdot x + 1(1 - x) = x^2 - x + 1$$

Allo stesso modo, per $p = 4$, otteniamo:

Percorsi	Traffico	Traffico ottimale
1-2	100%	66.87%
1-3-2	0%	33.13%

Costo totale	1
Costo ottimale	0.465
Prezzo dell'anarchia	2.1505

Anche in questo caso svolgendo i conti ($C(x) = x^5 - x + 1$) si ritrova esattamente la funzione di figura. La variazione di costo in un verso è più brusca e nell'altro più lenta. Uno spunto da poter cogliere sarebbe chiedersi quanto brusca possa diventare per sapere con quanta precisione dover stimare il flusso ottimale.

Raduniamo questi risultati in un unico grafico, in figura 7, dove confrontiamo i diversi prezzi dell'anarchia visti nella teoria rispetto all'esempio di Pigou con funzione di costo polinomiale sul primo ramo, cioè $1-p\cdot(p+1)^{-\frac{p+1}{p}}$ secondo [4], con i risultati numerici della simulazione.

Sulle ascisse troviamo l'esponente a cui è stato dato l'intervallo di variazione $0 \leq p \leq 4$. Sulle ordinate il prezzo dell'anarchia, diverso per ogni rete, sulla base di p. La funzione $1 - p \cdot (p + 1)^{-\frac{p+1}{p}}$ è rappresentata tramite linee spezzate mentre quella ottenuta dal calcolatore tramite punti. La corrispondenza è ottima e, avendo simulato l'instradamento di soli 1000 agenti, il risultato ottenuto è comunque molto preciso .

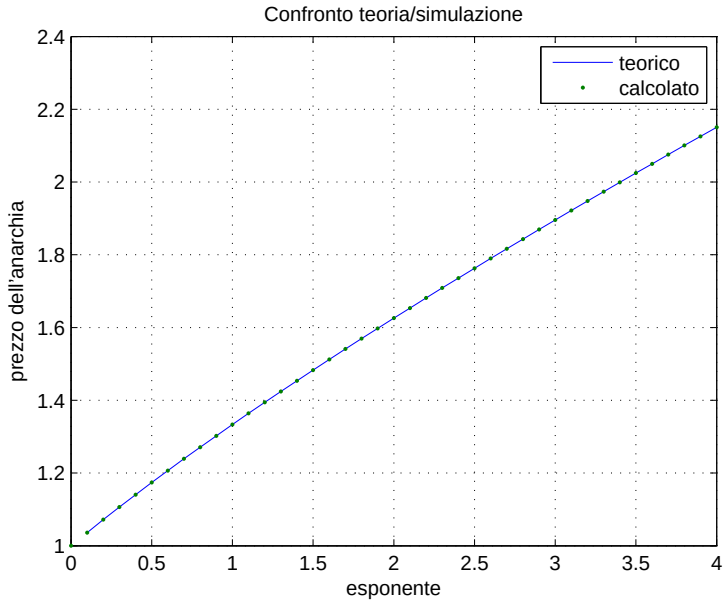

Figura 7: Prezzi dell'anarchia teorici e calcolati mediante MATLAB sull'esempio di Pigou con funzione di costo di esponente diverso.

Per la prossima simulazione modifichiamo il ramo di costo 1 con un ramo di costo $\frac{2}{3}$ in modo che da un certo punto in poi le scelte siano alternate e che non tutte optino per una stessa strada. Precisamente, i primi $\frac{2}{3}$ di traffico opteranno per la prima strada mentre ciò che rimane si divide.

Tracciamo il grafico di figura 8 simulando prima la partecipazione di 1000 agenti e, sopra, un secondo grafico che svolge la simulazione con soli 100 agenti.

Sulle ascisse c'è l'esponente p della funzione di costo $c(x) = x^p$ del primo percorso mentre sulle ordinate il prezzo dell'anarchia. Le 1000 unità di traffico reggono abbastanza bene (la funzione sembra abbastanza continua, così come dev'essere quella teorica). Riducendo le unità di traffico di un fattore 10 viene persa parecchia precisione. Si osserva che la non-atomicità è un'ipotesi da non poter scartare.

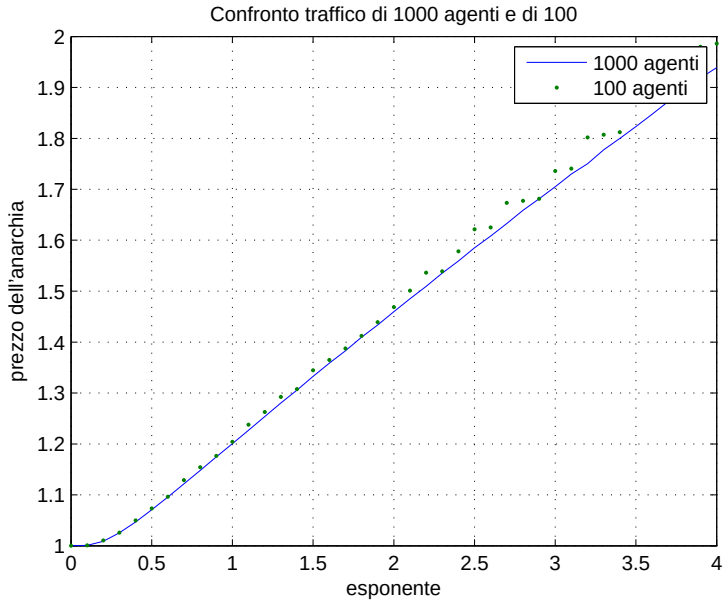

Figura 8: Prezzi dell'anarchia con e senza l'ipotesi di non-atomicità con funzione di costo di esponente diverso.

Riprendiamo ora il caso del paradosso Braess di figura 3 e facciamo notare che quanto osservato prima a parole si manifesta anche in pratica: la differenza di costo rispetto al flusso ottimale è del 33%.

In figura 9 sono riportati i diversi costi per ogni flusso nei 3 percorsi possibili.

Percorsi	Traffico	Traffico ottimale
1-2-3-4	100%	0%
1-2-4	0%	50%
1-3-4	0%	50%

Costo totale	2
Costo ottimale	1.5
Prezzo dell'anarchia	1.3333

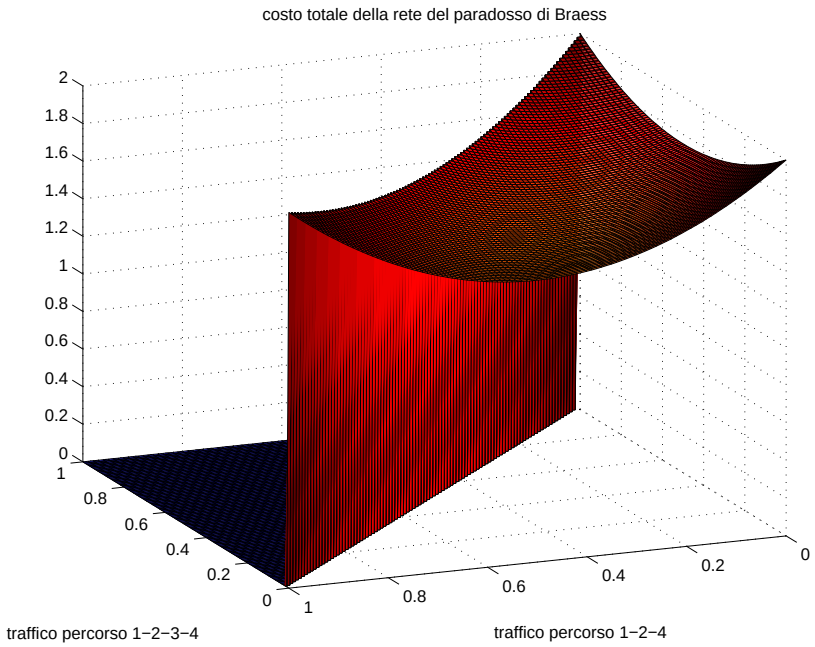

Figura 9: Grafico dei costi dei diversi flussi nel caso del paradosso di Braess.

Tutti gli esempi forniti finora hanno mostrato una buona utilità nel cercare il flusso ottimale ma ci si chiede ora se è solo un caso, se ci si possa aspettare spesso miglioramenti del genere o se in generale si ottenga anche di più. Purtroppo non è così: quelli visti finora sono esempi di reti semplici, con pochi gradi di libertà nella scelta del percorso. Nella pratica, in reti più complicate, i miglioramenti che si ottengono sono di bassa consistenza e il prezzo dell'anarchia spesso non è molto elevato.

Simuliamo allora un esempio di rete complicato, di figura 10, e andiamo a vedere di quanto risulta il miglioramento:

Assumiamo che il nodo di partenza sia il nodo 1 e quello di arrivo il nodo 3. L'algoritmo per la soluzione egoistica suggerisce l'uso del solo percorso 1-5-3 dei dieci disponibili. Anche il calcolo del flusso ottimale comporta di usare l'intera unità di traffico per lo stesso percorso. Come risultato si ha che il costo della soluzione egoistica è di 2, quello del flusso ottimale è anch'esso di 2 e il prezzo dell'anarchia risulta quindi 1.

I casi in cui il flusso derivante dalla disposizione egoistica degli agenti ed il flusso ottimale siano coincidenti non è affatto raro come suggerito da altri lavori [22].

Assumendo che sia stato un caso ripetiamo la simulazione cambiando il

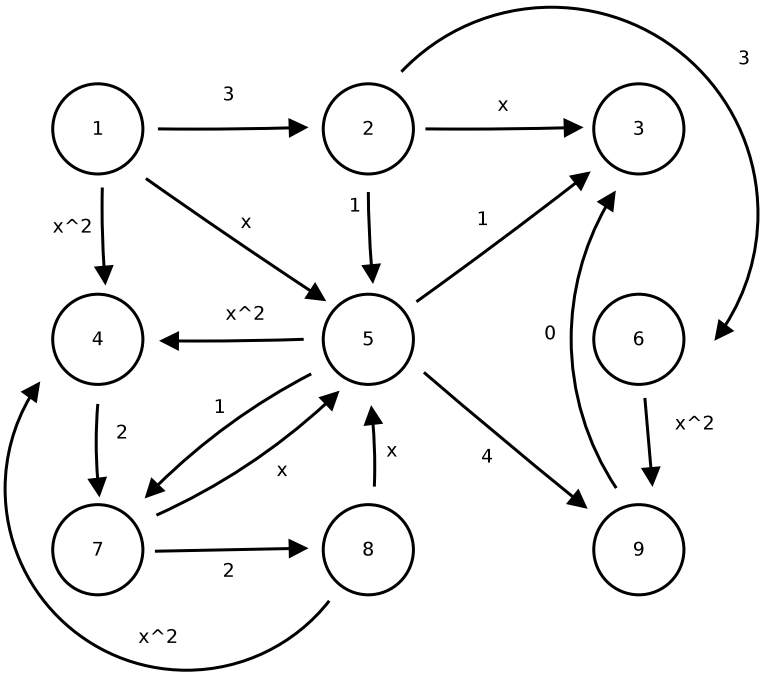

Figura 10: Esempio di rete complicata.

costo del ramo che va da 1 a 5 con uno di peso $c(x) = 2x$.

Mentre il traffico della scelta egoistica utilizza solo ed esclusivamente lo stesso percorso di prima, quello del flusso ottimale si divide in ben 3 percorsi dei 10 disponibili. Più precisamente, il 22% prende 1-2-3, il 17% prende 1-4-7-5-3e il 61% prende 1-5-3.

A questo punto i flussi sono molto differenti: quello ottimale è ben più distribuito nei diversi percorsi. Tuttavia il prezzo dell'anarchia non sale di molto rispetto a prima: il costo della soluzione egoistica è 3, quello del flusso ottimale è 2.6064 mentre il prezzo dell'anarchia è 1.1510. Sebbene buona parte del traffico sia in una posizione differente il vantaggio nel seguire il flusso ottimale non è molto.

Simuliamo ora una rete con funzioni di costo di carattere più forte, come gli esponenziali, i quali in breve tempo dovrebbero riuscire a congestionare i corrispettivi rami. L'esempio di rete proposto è in figura 11.

I risultati riferiti al nodo di arrivo 3 sono riportati in tabella.

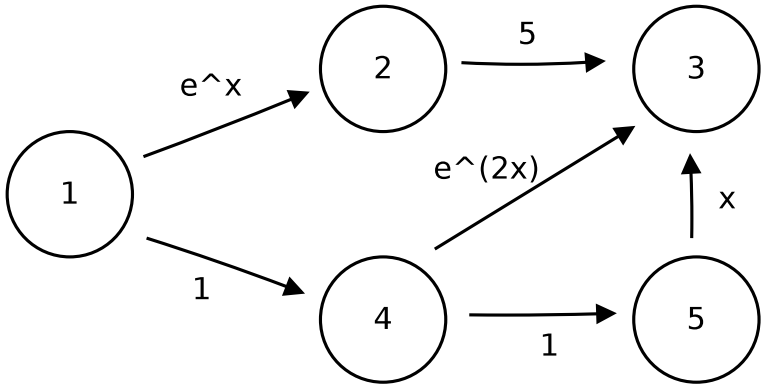

Figura 11: Esempio di rete con costi sui rami esponenziali.

Percorsi	Traffico	Traffico ottimale
1-2-3	0%	0%
1-4-3	28%	0.2526%
1-4-5-3	72%	0.7474%

Costo totale	2.7286
Costo ottimale	2.7246
Prezzo dell'anarchia	1.0014

Osserviamo che i costi sono molto simili perché il flusso egoistico non si è discostato di molto da quello ottimale.

In figura 12 è graficato come varia il costo totale rispetto al traffico sui rami.

Il traffico sul terzo ramo è il complemento a 1 della somma del traffico degli altri due. L'altezza del grafico rappresenta il costo totale secondo il flusso associato.

Dal grafico osserviamo che la variabilità di costo è alta come ci si aspettava dall'effetto degli esponenziali: il valore minimo e massimo sono all'incirca 3 e 8, l'uno meno della metà dell'altro. Notiamo invece che i due flussi individuati sono numericamente simili e il flusso generato egoisticamente si avvicina di molto a quello ottimale.

A proposito delle funzioni di costo è possibile fare qualche considerazione generale.

I rami esponenziali all'inizio crescono molto lentamente quindi è ragionevole per una buona fetta di agenti scegliere percorsi di questo tipo. Più lentamente cresce il costo di un percorso più sarà il numero di agenti per cui questo verrà scelto all'inizio.

Per mantenere un prezzo dell'anarchia basso quando ci si trova di fronte a funzioni di tipo diverso è necessario che il gap iniziale dei valori di queste

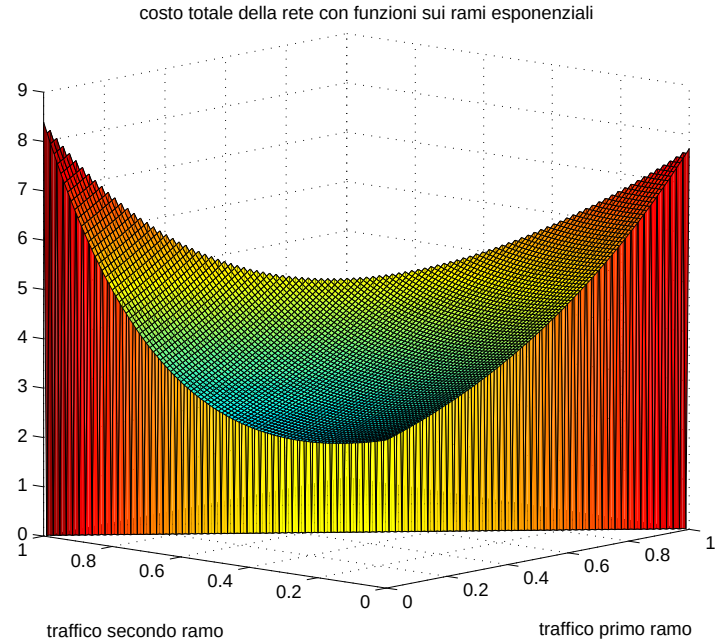

Figura 12: Grafico costo totale con funzioni sui rami esponenziali.

si consumi velocemente o non ci sia proprio. Ragioniamo infatti sugli esempi di Pigou e Braess con funzioni che sono quindi lineari o costanti. Le strade che, egoisticamente, gli agenti intraprendono sono socialmente sbagliate perché il gap iniziale è molto elevato: $c(x) = x$ all'inizio è nulla e solo dopo l'intera unità di traffico raggiunge il costo degli altri rami. Se avessimo avuto $c(x) = 1 + x$ il flusso scelto sarebbe coinciso con quello ottimale. La differenza tra i flussi egoistici e ottimali è quindi da attribuire alle funzioni di costo più rapide che, all'inizio, si presentano come molto più allettanti. Se inizialmente ogni percorso costasse uguale ad ogni altro la distribuzione degli agenti risulterebbe molto più accurata.

I risultati riscontrati scoraggiano l'uso di un controllo centralizzato perché raramente il prezzo dell'anarchia è abbastanza alto da valerne la pena in accordo a come viene fatto notare anche in altri lavori [22] [16].

5 Conclusioni e sviluppi futuri

La difficoltà principale resta il riuscire a trovare il flusso ottimale in un tempo accettabile. Quanto visto con MATLAB funziona ma è impensabile eseguire il toolbox di ottimizzazione ogni volta che un nodo sulla rete richiede di trasmettere dati ad un altro nodo e un algoritmo apposito per questo che abbia in uscita le indicazioni sulle quantità di traffico in un tempo ridotto è di difficile immaginazione. Se fosse applicato infatti su una rete molto grande dovrebbe trasmettere informazioni sui percorsi e i percorsi in reti molto grandi sono sequenze di nodi molto lunghe e ciò è tutto traffico aggiuntivo. È anche vero però che se il controllo è centralizzato le uniche informazioni da dover trasmettere ad ogni nodo sono, per ogni trasmissione che avviene in contemporanea, le percentuali di traffico della trasmissione da inviare ai nodi più vicini e quindi si risparmierebbe già qualcosa. Si potrebbe pensare altrimenti di applicarlo solo per la trasmissione di moli di dati consistenti ed evitare di rallentare la partenza di piccole trasmissioni, che sono quelle che avvengono più frequentemente.

In conclusione è interessante notare che sulle moderne reti di telecomunicazioni attuare tecniche centralizzate per ridurre i danni causati dall'instradamento egoistico non è sempre la soluzione migliore. Lasciare agire indipendentemente ogni nodo della rete come visto negli ultimi casi dei risultati porta ad avere quasi gli stessi rendimenti. In questi scenari la migliore soluzione tra le due è quella che comporta meno signaling. Tuttavia ci sono casi in cui la velocità ha un ruolo molto più rilevante: qualora scoppiasse un incendio in un edificio le uscite di sicurezza non possono correre il rischio di congestionarsi ed è chiaro che in situazioni di panico i soggetti sono più inclini ad intraprendere scelte egoistiche verso la direzione che appare in un primo momento la più veloce. Quindi una chiara indicazione da parte di una figura leader è necessaria.

I possibili sviluppi futuri sono osservare se quanto detto finora vale anche per i problemi di multi-commodity e qualificare in quali situazioni ricercare i flussi ottimali è una buona idea ma, prima di tutto, riuscire a trovare un algoritmo con adeguata precisione ma soprattutto buona velocità per il calcolo dei flussi ottimali nelle reti in generale.

Riferimenti bibliografici

[1] B. V. Cherkassky, A. V. Goldberg, T. Radzik, *Shortest paths algorithms: theory and experimental evaluation*. Mathematical Programming Series A, vol. 73, no. 2, pp. 129–174, 1996.

[2] L. Vangelista, *Introduction to telecommunication Services, Networks and Signaling*. In: N. Benvenuto, M. Zorzi, *Principles of Communications Networks and Systems*. Wiley, 2011.

[3] L. Badia, *Network Layers*. In: N. Benvenuto, M. Zorzi, *Principles of Communications Networks and Systems*. Wiley, 2011

[4] T. Roughgarden, *Selfish Routing and the Price of Anarchy*. Massachusetts Institute of Technology, 2005.

[5] A. C. Pigou, *The economics of welfare*. Macmillan and co., 1920.

[6] D. Braess, *Über ein Paradoxon aus der Verkehrsplanung*. Unternehmensforschung, 12:258-68, 1968.

[7] E. Koutsoupias, C. H. Papadimitriou, *Worst-case equilibria*, in Proc. STACS'99, pp. 404-413, 1999.

[8] E. W. Dijkstra, *A Note on Two Problems in Connexion with Graph*, Numerische Mathematik 1, pp. 269-271, 1959.

[9] R. Bellman, *On a routing problem*, Quarterly of Applied Mathematics 16, pp. 87–90, 1958.

[10] John Nash, *Non-Cooperative Games*, The annals of Mathematics, Second Series, vol. 54, no. 2, pp. 693, 1951.

[11] T. Roughgarden, Éva Tardos, *How bad is selfish routing?*. Journal of the ACM (JACM), vol. 49, no. 2, pp. 236-259, 2002.

[12] C. E. Perkins, E. M. Royer, *Ad-hoc on-demand distance vector routing*, in Proc. WMCSA '99, pp. 90-100, 1999.

[13] J. M. McQuillan, I. Richer, E. C. Rosen, *ARPANet Routing Algorithm Improvements*, BBN Report No. 3803, Cambridge, April 1978.

[14] P. Lauder, R. J. Kummerfeld, A. Fekete, *Hierarchical network routing*, in Proc. TRICOMM'91, pp 105–114, 1991.

[15] R. Feldmann, M. Gairing, T. Lücking, B. Monien, M. Rode, *Selfish Routing in Non-cooperative Networks: A Survey*. In: B. Rovan, P. Vojtáš, *Mathematical Foundations of Computer Science*. Springer Berlin/Heidelberg, 2003.

[16] L. Qiu, Y. R. Yang, Y. Zhang, S. Shenker, *On selfish routing in internet-like environments*. IEEE/ACM Transactions on Networking, vol. 14, no. 4, pp. 725-738, 2006.

[17] D. Fotakis, S. Kontogiannis, E. Koutsoupias, M. Mavronicolas, P. Spirakis, *The Structure and Complexity of Nash Equilibria for a Selfish Routing Game*. In: P. Widmayer, S. Eidenbenz, F. Triguero, R. Morales, R. Conejo, M. Hennessy, *Automata, Languages and Programming*. Springer Berlin/Heidelberg, 2002.

[18] R. Cole, Y. Dodis, T. Roughgarden, *How much can taxes help selfishrouting?*. Journal of Computer and System Sciences, vol. 73, no. 3, pp. 444-467, 2006.

[19] A. Czumaj, *Selfish routing on the Internet*. Handbook of scheduling: algorithms, models, and performance analysis, chap. 42, 2004.

[20] M. Gairing, B. Monien, K. Tiemann, *Selfish Routing with Incomplete Information*. Theory of computing systems, vol. 42, no. 1, pp.91-130, 2008.

[21] E. J. Friedman, *A generic Analysis of Selfish Routing*. 43rd IEEE Conference on Decision and Control, 2002.

[22] E. J. Friedman, *Genericity and congestion control in selfish routing*. 43rd IEEE conf. on Decision and Control, vol. 5, pp.4667-4672, 2004.

[23] A. Czumaj, B. Vöcking, *Tight bounds for worst-case equilibria*. ACM Transactions on Algorithms, vol. 3, no. 1, 2007.

[24] B. Wang, S. Soltani, J. K. Shapiro, P. N. Tan, *Local detection of selfish routing behavior in ad hoc networks*, in Proc. ISPAN 2005, Algorithms and Networks, pp. 392-399, 2005.

[25] T. Roughgarden, *Designing networks for selfish users is hard*, in Proc. FOCS 2001, pp. 472-481, 2001.

[26] S. Fischer, B. Vöcking, *On the Evolution of Selfish Routing*. In S. Albers, T. Radzik, *Algorithms*. Springer Berlin/Heidelberg, 2004.

[27] J. C. Harsanyi, *Games with Incomplete Information Played by Bayesian Players*. Management Science, vol. 14, no. 3 pp. 159-183, no. 5 pp. 320-334, no. 7 pp. 486-502, 1967.