Università degli Studi dell'Aquila

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

CORSO DI LAUREA IN INFORMATICA

TESI DI LAUREA TRIENNALE

Calcolo e performance di equilibri di Nash per il gioco della k-colorazione generalizzata

Relatore

Prof. Gianpiero Monaco

Laureando

Valentino Di Giosaffatte

ANNO ACCADEMICO 2017 - 2018

Indice

Parte I

Equilibri di Nash

Capitolo 1

Introduzione agli equilibri di Nash

1.1 Teoria dei giochi

La teoria dei giochi è la disciplina scientifica che si occupa dello studio e dell'analisi del comportamento e delle decisioni di soggetti razionali in un contesto di interdipendenza strategica.

Si definisce interdipendenza strategica, o interazione strategica, lo scenario in cui le decisioni di un individuo influenzano anche le scelte e gli scenari relativi agli altri individui.

Il principale oggetto di studio della teoria dei giochi sono le situazioni di conflitto nelle quali gli attori sono costretti ad intraprendere una strategia di competizione o cooperazione.

Tale scenario è definito gioco strategico e gli individui sono denominati giocatori.

Sulla base delle premesse e delle regole compositive del gioco in oggetto, viene costruito un modello matematico nel quale ciascun giocatore effettua le proprie decisioni (mosse migliorative) seguendo una strategia finalizzata ad aumentare il proprio vantaggio netto.
A ciascuna scelta positiva corrisponde un ritorno favorevole in termini di beneficio (payoff), il medesimo concetto vale in modo contrario in caso di scelta negativa, in tal caso il ritorno sarà sfavorevole.

In tali scenari le decisioni di un soggetto possono influire direttamente sui risultati conseguibili dagli altri e viceversa secondo un meccanismo di retroazione.

La teoria dei giochi è un concetto di soluzione applicabile ad un'ingente molteplicità di casi nei quali una pluralità di agenti decisionali possono operare in maniere competitiva, seguendo interessi contrastanti, o in maniera cooperativa, seguendo l'interesse comune.

1.1.1 Giochi non-cooperativi

In questo documento, la trattazione sarà incentrata sull'analisi di una particolare tipologia di giochi : i giochi non-cooperativi.

I giochi non-cooperativi, detti anche competitivi, rappresentano una specifica classe di giochi nella quale i giocatori non possono stipulare accordi vincolanti di cooperazione (anche normativamente), indipendentemente dai loro obiettivi.

Il criterio di comportamento razionale adottato nei giochi non-cooperativi è di carattere individuale ed è denominato strategia del massimo.

La suddetta definizione di razionalità va a modellare il comportamento di un individuo intelligente e ottimista che si prefigge l'obiettivo di prendere sempre la decisione che consegue il massimo guadagno possibile, perseguendo di conseguenza sempre la strategia più vantaggiosa per se stesso.

Si parla dunque di punto di equilibrio qualora nel gioco esista una strategia che presenti il massimo guadagno per tutti i giocatori, ovvero uno stato stabile del gioco nel quale tutti gli attori ottengono il massimo profitto individuale e collettivo.

1.2 Equilibri di Nash

La precedente affermazione muove l'oggetto della trattazione verso l'argomento centrale di questo studio, ovvero gli equilibri di Nash.

L'equilibrio di Nash è una combinazione di strategie nella quale ciascun giocatore effettua la migliore scelta possibile, seguendo cioè una strategia dominante, sulla base delle aspettative di scelta dell'altro giocatore.

L'equilibrio di Nash è la combinazione di mosse (m1, m2) in cui la mossa di ciascun giocatore è la migliore risposta alla mossa effettuata da un altro giocatore.

Ciascun giocatore formula delle aspettative sulla scelta dell'altro giocatore e in base a queste decide la propria strategia, con l'obiettivo di massimizzare il proprio profitto e di conseguenza quello degli altri.

Un equilibrio di Nash è un equilibrio stabile, poiché nessun giocatore ha interesse a modificare la propria strategia.

Ciascun giocatore trae la massima utilità possibile dalle proprie scelte, tenendo conto della migliore scelta dell'altro giocatore, e dunque qualunque variazione alla propria strategia potrebbe soltanto peggiorare il proprio valore di tornaconto (payoff o utilità).

L'equilibrio di Nash è conosciuto anche con il nome di equilibrio non cooperativo poiché rappresenta una situazione di equilibrio ottimale per un gioco non-cooperativo.

L'equilibrio di Nash non deriva dall'accordo tra i giocatori, bensì dall'adozione di strategie dominanti perseguite da tutti i giocatori, tali da garantire sia il miglior profitto possibile per ciascun giocatore (ottimo individuale), sia il miglior equilibrio collettivo (ottimo sociale).

1.2.1 Definizione formale

Definiamo ora alcuni concetti basilari e chiariamo alcuni aspetti matematici della teoria dei giochi al fine di delineare in modo più accurato il concetto di equilibrio di Nash.

Un gioco è caratterizzato da :

- un insieme G di giocatori, o agenti, che indicheremo con $i = 1, \ldots, N$
- un insieme S di strategie, costituito da un insieme di M vettori

  $$S_i = (s_{i,1}, s_{i,2}, \ldots, s_{i,j}, \ldots, s_{i,M_i})$$

  ciascuno dei quali contiene l'insieme l'insieme delle strategie che il giocatore i-esimo ha a disposizione, cioè l'insieme delle azioni che esso può compiere.
  (indichiamo con s_i la strategia scelta dal giocatore i)
- un insieme U di funzioni

  $$u_i = U_i (s_1, s_2, \ldots, s_i, \ldots, s_N)$$

che associano ad ogni giocatore i il guadagno (detto anche payoff) u_i derivante da una data combinazione di strategie (il guadagno di un giocatore in generale non dipende solo dalla propria strategia ma anche dalle strategie scelte dagli avversari)

Un equilibrio di Nash per un dato gioco è una combinazione di strategie (che indichiamo con l'apice e)

$$s_1^e, s_2^e, ..., s_N^e$$

tale che

$$U_i\left(s_1^e, s_2^e, ..., s_i^e, ..., s_N^e\right) \geq U_i\left(s_1^e, s_2^e, ..., s_i, ..., s_N^e\right)$$

$\forall i$ e $\forall s_i$ scelta dal giocatore i-esimo.

Il significato di quest'ultima disuguaglianza è il seguente : se un gioco ammette almeno un equilibrio di Nash, ciascun agente ha a disposizione almeno una strategia s_i^e dalla quale non ha alcun interesse ad allontanarsi se tutti gli altri giocatori hanno giocato la propria strategia s_j^e.

Come si può facilmente desumere direttamente dalla suddetta disequazione, se il giocatore i gioca una qualunque strategia a sua disposizione diversa da s_i^e, mentre tutti gli altri giocatori hanno giocato la propria strategia s_j^e, può solo peggiorare il proprio guadagno o, al più, lasciarlo invariato.

Da qui si può dedurre quindi che se i giocatori raggiungono un equilibrio di Nash, nessuno può più migliorare il proprio risultato modificando solo la propria strategia, ed è quindi vincolato alle scelte degli altri.

Poiché questo vale per tutti i giocatori, è evidente che se esiste un equilibrio di Nash ed è unico, esso rappresenta la soluzione del gioco, in quanto nessuno dei giocatori ha interesse a cambiare strategia.

1.3 Nozioni di base

1.3.1 Rappresentazione con matrici di payoff e descrizione del procedimento decisionale

Nella seguente matrice di payoff è rappresentato un esempio di equilibrio di Nash in un gioco non-cooperativo a 2 giocatori (tale equilibrio può essere esteso a N giocatori).

		G2	
		S1	**S2**
G1	**S1**	A [2, 2]	B [2, 1]
	S2	C [1, 2]	D [1, 1]

Tabella 1.1: Matrice di payoff

Nella suddetta matrice ciascun giocatore può scegliere la strategia S1 o la strategia S2. Il giocatore 1 si aspetta che il giocatore 2 scelga S1 (strategia dominante) e, quindi, adotta anch'egli la strategia S1 in quanto gli consente di ottenere un payoff individuale pari a 2.

Anche il giocatore 2 formula delle aspettative sulle scelte dell'avversario e si attende che il giocatore 1 scelga S1, scegliendo a sua volta la strategia S1.

L'equilibrio del gioco converge verso la cella A della matrice nella quale entrambi i giocatori massimizzano il proprio payoff individuale (ottimo individuale) dopo aver scelto la propria strategia dominante. Entrambi i giocatori hanno formulato un'ipotesi sulla migliore scelta del giocatore avversario e, sulla base di questa, hanno scelto la propria strategia dominante.

Cerchiamo ora di comprendere al meglio il funzionamento del processo decisionale alla base dell'equilibrio di Nash.

		G2	
		S1	**S2**
G1	**S1**	A [2, 2]	B [1, 1]
	S2	C [1, 2]	D [2, 1]

Tabella 1.2: Processo decisionale

Il giocatore 1 potrebbe scegliere sia S1 che S2, in entrambi i casi potrebbe sperare di ottenere per sé il payoff più alto (2) nelle celle A e D. Il giocatore 1 sa bene però che se scegliesse S2, il giocatore 2 sceglierebbe in seguito S1 e l'equilibrio finale si collocherebbe nella cella C, nella quale egli otterrebbe il payoff più basso (1).

Qualora scegliesse invece S1, il giocatore 1 sarebbe consapevole che anche il giocatore 2 sceglierebbe S1 e l'equilibrio finale in questo caso si collocherebbe nella cella A, nella quale il giocatore 1 otterrebbe il payoff più alto (2).

Seguendo il medesimo ragionamento, qualora spettasse al giocatore 2 scegliere per primo, questi sarebbe consapevole che scegliendo S1 anche il giocatore 1 sceglierebbe S1. Dunque anche il questo caso l'equilibrio strategico si collocherebbe nella cella A.

Il giocatore 2 non sceglierebbe mai S2 in quanto in ogni caso avrebbe il payoff più basso (1).

La cella A è un equilibrio di Nash, tutte le altre non lo sono.

1.3.2 Equilibri di Nash e ottimo sociale

		G2	
		S1	**S2**
G1	**S1**	A [2, 2]	B [2, 1]
	S2	C [1, 2]	D [1, 1]

Tabella 1.3: Ottimo sociale

Nell'esempio 1.1 appena trattato possiamo inoltre constatare facilmente che l'equilibrio di Nash trovato (cella A della matrice) è anche un ottimo sociale. Nel suddetto equilibrio coesistono una situazione ottimale per entrambi i giocatori (entrambi i giocatori possiedono un payoff massimo) e una situazione ottimale per l'intera collettività (in quanto la somma dei valori di payoff di entrambi i giocatori è uguale 4, il valore maggiore all'interno della matrice)

1.3.3 Equilibri di Nash multipli

Specifichiamo inoltre che un gioco non-cooperativo può presentare più equilibri di Nash. Anche il presenza di equilibri multipli, ciascun equilibrio è comunque stabile, poiché dalla propria posizione di equilibrio locale qualsiasi scelta è peggiorativa per ogni giocatore.

Al fine di verificare quest'ultima affermazione viene presentata la seguente matrice di payoff nella quale sono presenti 2 equilibri di Nash simmetrici (nelle celle A e D).

		G2	
		S1	**S2**
G1	**S1**	A [2, 2]	B [2, 1]
	S2	C [1, 2]	D [2, 2]

Tabella 1.4: Equilibri di Nash multipli

1.3.4 Assenza di equilibri di Nash

In alcuni casi possono essere del tutto assenti le condizioni per determinare un equilibrio di Nash e di conseguenza molti giochi sono privi di equilibrio. Per verificare tale asserzione mostriamo la matrice di payoff relativa ad un gioco nel quale non è possibile trovare un equilibrio di Nash.

		G2	
		S1	**S2**
G1	**S1**	A [1, 1]	B [1, 0]
	S2	C [2, 1]	D [0, 2]

Tabella 1.5: Assenza di equilibri di Nash

Nell'esempio appena descritto, se il giocatore 1 sceglie la strategia S2 (tentando di ottenere il payoff 2), il giocatore 2 sceglierà la strategia S2 (payoff 2) e l'equilibrio si posiziona nella cella D della matrice.

Se al contrario è il giocatore 2 a scegliere la strategia S2 (tentando a sua volta di ottenere il payoff 2), il giocatore 1 sceglierà la strategia S1 (payoff 1) e l'equilibrio si posiziona nella cella B della matrice. Nel suddetto esempio non esiste dunque un equilibrio di Nash.

1.3.5 Il dilemma del prigioniero : sub-ottimalità individuale e sociale

L'adozione di strategie dominanti non assicura sempre un equilibrio di ottimo sociale. In assenza di informazioni, un gioco non-cooperativo potrebbe convergere verso un equilibrio stabile ma non ottimale.

L'ipotesi è dimostrata nel problema del "dilemma del prigioniero" in cui due attori, pur formulando delle aspettative razionali sul comportamento dell'avversario e adottando strategie dominanti, determinano un equilibrio sub-ottimale (D) sia dal punto individuale che sociale.

La seguente matrice payoff evidenzia il tipico caso del dilemma del prigioniero : vi sono 2 giocatori separati in stanze differenti, senza che abbiano la possibilità di comunicare (informazione imperfetta) ; i 2 giocatori sono accusati di un reato e interrogati contemporaneamente.

- il giocatore che confessa il reato accusando l'altro ottiene la scarcerazione (utilità 9), mentre il giocatore accusato subisce il massimo della pena (utilità 0)
- se entrambi i giocatori evitano di confessare, ai 2 viene applicata un pena molto lieve (utilità 5)
- se entrambi i giocatori confessano, vengono condannati alla pena ordinaria (utilità 2)

		G2	
		non confessa	**confessa**
G1	**non confessa**	A [5, 5]	B [0, 9]
	confessa	C [9, 0]	D [2, 2]

Tabella 1.6: Il dilemma del prigioniero

Nell'esempio il giocatore 1 si aspetta che il giocatore 2 confessi, poiché la strategia dominante del giocatore 2 è confessare, grazie alla confessione quest'ultimo otterrebbe la libertà (payoff 9).

Sulla base di questa aspettativa il giocatore 1 sceglie la propria strategia dominante tra B e D, scegliendo a sua volta di confessare per ottenere un payoff uguale a 2. Lo stesso ragionamento viene adottato, in modo inverso, dal giocatore 2.

L'equilibrio D è un equilibrio di Nash ed è un equilibrio stabile poiché nessuno dei 2 giocatori ha interessa a modificare le proprie scelte.

In tale equilibrio però entrambi i giocatori ottengono un payoff subottimale pari a 2 rispetto a quello che avrebbero se decidessero entrambi di non confessare (payoff 5 - equilibrio A) e in più l'equilibrio D presenta un valore di ottimo sociale (4) inferiore rispetto a quello che si potrebbe ottenere dall'equilibrio A (10) che rappresenta l'ottimo individuale e sociale del gioco.

In conclusione, il dilemma del prigioniero rappresenta una situazione in cui le scelte individuali del giocatori, pur essendo strategie dominanti, determinano un equilibrio inefficiente. Nel dilemma del prigioniero l'equilibrio del gioco non è né un ottimo individuale né un ottimo sociale.

Parte II

Gioco della K-Colorazione Generalizzata

Capitolo 2

Gioco della k-colorazione generalizzata

Esaminiamo ora gli equilibri di Nash puri per il gioco della k-colorazione generalizzata nel quale viene fornito un grafo orientato e un insieme di k colori.

I nodi rappresentano i giocatori e gli archi catturano i loro reciproci interessi.

La strategia di ciascun giocatore è composta da k colori.

L'utilità di un giocatore v in un dato stato o colorazione è data dalla somma dei pesi degli archi (v, u) incidenti a v tale che il colore scelto da v sia diverso da quello scelto da u, più il profitto guadagnato dall'utilizzo del colore scelto.

Per prima cosa dimostriamo che il gioco della k-colorazione generalizzata è convergente e dunque esiste sempre almeno un equilibrio di Nash per ogni istanza del gioco in questione.

Valutiamo dunque in seguito una descrizione delle prestazioni dei giochi della k-colorazione generalizzata per mezzo delle nozioni largamente utilizzate di prezzo dell'anarchia (price of anarchy) e prezzo della stabilità (price of stability).

Forniamo inoltre limiti stretti per 2 tipi di benessere sociale ampiamente utilizzati, il benessere sociale utilitario (utilitarian social welfare) e il benessere sociale egalitario (egalitarian social welfare).

2.1 Descrizione generale

Le istanze appartenenti al gioco della k-colorazione generalizzata sono giocati su grafi non-orientati pesati in cui i nodi corrispondono ai giocati e in cui gli archi identificano le connessioni sociali o le relazioni tra i giocatori.

Il set di strategie di ciascun giocatore è un insieme di k colori disponibili (assumiamo che i colori siano gli stessi per ogni giocatore).

Quando i giocatori selezionano un colore inducono una colorazione k (o semplicemente una colorazione).

Ciascun giocatore possiede una funzione di profitto legata all'apprezzamento da parte di quest'ultimo del colore scelto (vale per tutti i colori disponibili per il giocatore).

Data una colorazione, l'utilità (o il guadagno) di un giocatore v colorato con il colore i è la somma dei pesi degli archi (v, u) incidenti a v, tale che il colore scelto da v è diverso da quello scelto da u, più il profitto derivante dalla scelta del colore i da parte del giocatore v.

Assumiamo che i giocatori siano egoisti, dunque un concetto di soluzione ben noto per questo tipo di impostazione è l'equilibrio di Nash.

L'equilibrio di Nash è uno dei concetti più importanti nella teoria dei giochi e fornisce una soluzione stabile che è robusta alle deviazioni dei singoli giocatori.

Formalmente, una colorazione è un equilibrio di Nash puro se nessun giocatore può migliorare la propria utilità deviando unilateralmente dalla propria strategia attuale.

L'egoismo dei giocatori può causare in molti casi la perdita di benessere sociale e quindi una soluzione stabile non è sempre buona rispetto al benessere della società.

Consideriamo ora 2 nozioni di benessere sociale, naturali e ampiamente utilizzate.

Data una colorazione k, il benessere sociale utilitario (utilitarian social welfare) è definito come la somma delle utilità dei giocatori nella colorazione k, mentre il benessere sociale egalitario (egalitarian sociale welfare) è definito come l'utilità minima tra tutti i giocatori nella colorazione k.

Utilizziamo inoltre 2 metodi per misurare la bontà di un equilibrio di Nash rispetto a un benessere sociale, il prezzo dell'anarchia (price of anarchy) e il prezzo della stabilità (price of stability).

Il prezzo dell'anarchia descrive, nel peggiore dei casi, come l'efficienza di un sistema degrada a causa del comportamento egoistico dei suoi giocatori, mentre il prezzo della stabilità ha un naturale significato di stabilità, poiché è la soluzione ottimale tra quelle che possono essere accettate da giocatori egoisti.

Studiamo ora l'esistenza e le performance degli equilibri di Nash nei giochi della k-colorazione generalizzata.

Ci concentriamo solo sui grafi non-orientati poiché per i grafi orientati anche il problema di decidere se un'istanza ammetta un equilibrio di Nash è un problema difficile (NP-Hard), inoltre esistono casi per i quali un equilibrio di Nash non esiste affatto.

2.1.1 Nozioni sul problema

Sappiamo che in caso di grafi non-orientati non-pesati è possibile calcolare un equilibrio di Nash in tempo polinomiale.

Nel nostro caso, il problema di calcolare un equilibrio di Nash su grafi non-orientati pesati è PLS-Completo anche per $k = 2$, dato che il gioco del taglio massimo (Max-cut game) è un caso speciale del nostro gioco.

Proprio riguardo questo aspetto è bene delineare la relazione che esiste tra il gioco della k-colorazione generalizzata e il gioco del taglio massimo, un problema molto importante e ampiamente trattato in letteratura.

Il gioco della k-colorazione generalizzata è un estensione del gioco del taglio massimo, infatti quest'ultimo può essere ottenuto ponendo a 0 i profitti relativi ai colori e ponendo a 2 il numero di colori presenti nel set disponibile per ciascun giocatore.

Inoltre il gioco della k-colorazione generalizzata è un'estensione del gioco della k-colorazione nel quale vi sono k-colori ma i profitti relativi ai colori sono impostati a 0.

2.2 Dettagli sul modello

Dato un grafo semplice non-orientato $G = (V, E, w)$, dove $|V| = n$, $|E| = m$ e $w : E \to \mathbb{R}_{\geq 0}$ è la funzione per i pesi sugli archi che associa un peso positivo a ciascun arco.

Denotiamo con $\delta^v(G) = \sum_{u \in V : \{v,u\} \in E} w(\{v, u\})$ la somma dei pesi di tutti gli archi incidenti a v.

L'insieme dei nodi con cui un nodo v ha un arco in comune è chiamato insieme dei vicini di v (insieme dei nodi adiacenti a v).

Un'istanza di gioco della k-colorazione generalizzata è un tupla (G, K, P). $G = (V, E, w)$ è un grafo pesato non-orientato senza self loops, in cui ogni $v \in V$ è un giocatore egoista.

K è un insieme di colori disponibili (assumiamo $K \geq 0$).

Il set di strategia di ciascun giocatore è dato dai k colori disponibili, ovvero i giocatori hanno lo stesso insieme di azioni.

Denotiamo con $P : V \times K \to \mathbb{R}_{\geq 0}$ la funzione di profitto del colore, che definisce quanto un giocatore apprezza un colore, ovvero se il giocatore v scegli di usare il colore i, allora guadagna $P_v(i)$.

Per ciascun giocatore v, definiamo P_v^M come il massimo profitto che v può guadagnare da un colore, formalmente $P_v^M = max_{i=1,...,k} P_v(i)$.

Quando $P_v(i) = 0 \forall v \in V$ e $\forall i \in k$, si ha il caso in cui non vi sono profitti associati ai colori scelti, quindi possiamo riferirci a questo gioco come a un gioco della k-colorazione (graph k-coloring game).

Uno stato del gioco $c = \{c_1, \ldots, c_n\}$ è una k-colorazione, o semplicemente una colorazione, dove c_v è il colore (cioè un numero $1 \leq c_v \leq k$) scelto dal giocatore v.

In una determinata colorazione c, il payoff (o l'utilità) di un giocatore v è la somma dei pesi degli archi (v, u) incidenti a v, tale che il colore scelto da v è diverso da quello scelto da u, oltre al profitto ottenuto dall'utilizzo il colore scelto.

In modo formale, per una colorazione c, il payoff di un giocatore v è $\mu_c(v) = \sum_{u \in V : \{v,u\} \in E \wedge c_v \neq c_u} w(\{v, u\}) + P_v(c_v)$.

Quando un arco (v, u) fornisce utilità ai suoi endpoints in una colorazione c, cioè quando $c_v \neq c_u$ diciamo che tale arco è corretto.

diciamo anche che un arco (v, u) è monocromatico in una colorazione c quanto $c_v = c_u$.

Sia c_{-v}, c'_u la colorazione ottenuta da c cambiando la strategia del giocatore v da c_v a c'_v.

Data una colorazione $c = \{c_1, \ldots, c_n\}$, una mossa migliorativa (improving move) del giocatore v nella colorazione c è una strategia c'_v tale che $\mu_{(c_{-v}, c'_v)}(v) > \mu_c(v)$.

Uno stato del gioco è un equilibrio di Nash puro o equilibrio stabile se e so se nessun giocatore può effettuare una mossa migliorativa.

In modo formale, $c = \{c_1, \ldots, c_n\}$ è un equilibrio di Nash se $\mu_c(v) \geq \mu_{(c_{-v}, c'_v)}(v)$ per ogni possibile colorazione c'_v e per ogni giocatore $v \in V$.

Una dinamica di miglioramento (improving dynamic), o brevemente dinamica (dynamic), è una sequenza di mosse migliorative. Si dice che un gioco sia convergente se, dato un qualsiasi stato iniziale c, qualsiasi sequenza di mosse migliorative porta a un equilibrio di Nash.

Data una colorazione c, definiamo una funzione di benessere sociale utilitario (utilitarian social welfare) $SW_{UT}(c)$ e una funzione di benessere sociale egalitario (egalitarian sociale welfare) $SW_{EG}(c)$ come segue :

$$SW_{UT}(c) = \sum_{v \in V} \mu_c(v) = \sum_{v \in V} P_v(c_v) + \sum_{\{v,u\} \in E : c_v \neq c_u} 2w(\{v, u\})$$

$$SW_{EG}(c) = min_{v \in V} \mu_c(v)$$

Indichiamo con C l'insieme di tutte le possibili colorazioni e denotiamo con Q l'insieme di tutte le colorazioni stabili.

Data una funzione di benessere sociale SW, definiamo il prezzo dell'anarchia (price of anarchy) (PoA) per il gioco della k-colorazione generalizzata come il rapporto tra il massimo benessere sociale tra tutte le possibili colorazioni sul minimo benessere sociale tra tutte le possibili colorazioni stabili.

In modo formale, $PoA = \frac{max_{c \in C} SW(c)}{min_{c' \in Q} SW(c')}$.

Definiamo inoltre il prezzo della stabilità (price of stability) (PoS) per il gioco della k-colorazione generalizzata come il rapporto tra il massimo be-

nessere sociale tra tutte le possibili colorazioni sul massimo benessere sociale tra tutte le possibili colorazioni stabili.
In modo formale, $PoS = \frac{max_{c \in C} SW(c)}{max_{c' \in Q} SW(c')}$.

Intuitivamente, il PoA (rispettivamente PoS) ci dice quanto è peggiore il benessere sociale nel peggiore (rispettivamente migliore) equilibrio di Nash, relativo al benessere sociale dell'ottimo.

2.2.1 Convergenza ed esistenza degli equilibri di Nash

Per prima cosa mostriamo che il gioco della k-colorazione generalizzata è convergente. Ciò implica chiaramente che gli equilibri di Nash esistono sempre.

Proposizione 1. $\forall k$, *ogni gioco della k-colorazione generalizzata* (G, K, P) *finito è convergente.*

Notiamo che, da un lato, se il grafo non è pesato, la dinamica, partendo dalla colorazione in cui ogni giocatore v selezione il colore in modo tale da ottenere il massimo profitto possibile, che è, il colore i tale che $P_v(i) = P_v^M$, converge ad un equilibrio di Nash in al massimo $|E|$ mosse migliorative.

D'altra parte, se il grafo è pesato, il calcolo di un equilibrio di Nash è PLS-Completo.

Ne consegue il fatto che, quando $k = 2$, il nostro gioco è una generalizzazione del gioco del taglio (cut game) che è uno dei primi problemi che si sono dimostrati essere PLS-Completi.

2.3 Benessere sociale utilitario (utilitarian social welfare)

In questa sezione ci concentreremo sul benessere sociale utilitario. Mostriamo limiti stretti per il prezzo dell'anarchia utilitario e tralasciamo quelli per il prezzo della stabilità utilitario.

2.3.1 Prezzo dell'anarchia utilitario (utilitarian price of anarchy)

Ricordiamo che nel caso senza profitti associati ai colori, il prezzo dell'anarchia utilitario è esattamente $\frac{k}{(k-1)}$.

Qui dimostriamo che il gioco della k-colorazione generalizzata il prezzo dell'anarchia utilitario è pari a 2, cioè indipendente dal numero di colori.

Iniziamo dimostrando che il prezzo dell'anarchia utilitario è al più 2.

Teorema 1. *Il prezzo dell'anarchia per il gioco della k-colorazione generalizzata è al più 2.*

Mostriamo ora che il prezzo dell'anarchia utilitario è almeno 2 anche per il caso speciale di grafi stella non-pesati.

Teorema 2. *Il prezzo dell'anarchia utilitario per il gioco della k-colorazione generalizzata è almeno 2, anche per il caso speciale di grafi stella non-pesati.*

2.4 Benessere sociale egalitario (egalitarian social welfare)

In questa sezione ci concentriamo sul benessere sociale egalitario. Mostriamo limiti stretti per il prezzo dell'anarchia egalitario e tralasciamo quelli per il prezzo della stabilità egalitario.

2.4.1 Prezzo dell'anarchia egalitario (egalitarian price of anarchy)

Per un gioco della k-colorazione (ovvero senza profitti associati ai colori), un limite inferiore per il prezzo dell'anarchia egalitario è fornito.

In tal caso, la soluzione ottimale è tale che ciascun giocatore v ha un'utilità uguale al suo grado δ^v, tuttavia esiste una colorazione stabile in cui ciascun giocatore v ha un'utilità pari a $\frac{(k-1)}{k}\delta^v$, per qualsiasi numero di colori $k \geq 2$.

Questo risultato insieme al fatto che secondo il pidgehole principle, per qualsiasi colorazione stabile ogni giocatore v raggiunge almeno un'utilità pari a $\frac{(k-1)}{k}\delta^v$, implica che il prezzo dell'anarchia egalitario è esattamente

$\frac{(k-1)}{k}$ per il gioco della k-colorazione generalizzata.

Pertanto ora consideriamo il gioco della k-colorazione generalizzata. Per prima cosa notiamo che l'istanza definita nel Teorema 2 fornisce un limite inferiore di 2 per il prezzo dell'anarchia egalitario.

Infatti, nella colorazione ottimale l'utilità minima è 2 ed esiste una colorazione stabile in cui l'utilità minima è 1.

Inoltre, si sottolinea che la dimostrazione del Teorema 1 mostra sostanzialmente che per ogni giocatore i, la propria utilità in un qualsiasi risultato stabile è almeno la metà del profitto che ha nell'ottimo.

Quindi otteniamo facilmente il seguente teorema relativo al prezzo dell'anarchia egalitario per il gioco della k-colorazione generalizzata.

Teorema 3. *Il prezzo dell'anarchia egalitario per il gioco della k-colorazione generalizzata è 2.*

Parte III

Implementazione

Capitolo 3

Implementazione

In questo capitolo andremo a delineare e descrivere gli aspetti fondamentali ed essenziali relativi all'attività di implementazione.

Inizieremo con una presentazione della struttura generale dei programmi assieme ad una descrizione accurata dei componenti utilizzati e delle scelte effettuate in fase di progettazione.

In seguito procederemo con un'attenta analisi degli algoritmi implementati, ovvero il l'algoritmo per il calcolo degli equilibri di Nash, quello per il calcolo dell'ottimo relativo alla funzione di benessere sociale utilitario e quello per il calcolo dell'ottimo relativo alla funzione di benessere sociale egalitario.
Tale sezione sarà caratterizzata dall'utilizzo di pseudocodice per ciascuno degli algoritmi in modo tale da rendere più comprensibile la descrizione dei cicli e delle operazioni.

Tratteremo in modo approfondito questa porzione del documento poiché precede la sezione relativa alla sperimentazione effettuata attraverso i programmi implementati e dunque è di fondamentale importanza.

3.1 Struttura generale

Procediamo presentando la struttura generale relativa programmi implementati.
Quest'ultima è rappresentata attraverso una struttura ad albero che riproduce una porzione di filesystem partendo dalla root del progetto.
Al fine di rendere più chiara la lettura viene inoltre fornita un breve leggenda sulla nomenclatura utilizzata.

- La nomenclatura **nome.dir** indica che l'oggetto è una cartella

- La nomenclatura **nome.edgelist** indica che l'oggetto è un file con estensione .edgelist (oggetto principale modellato dal programma)
- La nomenclatura **nome.dot** indica che l'oggetto è un file con estensione .dot (oggetto utilizzato su macchine GNU/Linux per il disegno attraverso la libreria PyGraphViz in fase di debug)
- le lettere **X,K,Y,Z,H,W** rappresentano numeri casuali (sono utilizzate per descrivere la moltitudine di cartelle, grafi creati e risultati, ottenuti durante un generale caso d'uso dei programmi)
- la nomenclatura **nome.init** indica che l'oggetto è un file con estensione .init (oggetto utilizzato in fase di lettura per salvare le caratteristiche dei grafi (nodi, archi, pesi, colorazione) e dei colori (colori, profitti))
- la nomenclatura **nome.out** indica che l'oggetto è un file con estensione .out (oggetto utilizzato in fase di lettura per salvare i risultati derivanti da esecuzioni singole o multiple usando gli algoritmi per il calcolo del nash, dell'ottimo con funzione di benessere sociale utilitario e dell'ottimo con funzione di benessere sociale egalitario)

Descriviamo ora la funzione basilare di alcuni componenti dell'albero sottostante rappresentante un esempio generale relativo ad un caso d'uso dei programmi.

- La cartella **generator** contiene al suo interno l'intera struttura relativa al generatore di grafi
- La cartella **gen** contiene i risultati delle generazioni singole di grafi
  1. Al suo interno vi sono X cartelle gen-dir-X (il nome viene assegnato dinamicamente dall'utente) generate dinamicamente
  2. Ciascuna cartella contiene il risultato di una generazione singola di un grafo sotto forma di una coppia di file .edgelist e .dot (utilizzato solo in fase di debug)
  3. Ogni gruppo cartella-file.edgelist-file.dot rappresenta il risultato di una generazione singola
- La cartella **m-gen** contiene i risultati delle generazioni multiple di grafi
  1. Al suo interno vi sono Y cartelle m-gen-dir-Y (il nome viene assegnato dinamicamente dall'utente) generate dinamicamente
  2. Ciascuna cartella contiene il risultato di una generazione multipla di grafi sotto forma di molteplici coppie (nell'esempio K,Z,...) di file .edgelist e .dot (utilizzato solo in fase di debug)

3. Ogni gruppo cartella-file.edgelist-file.dot-file.edgelist-file.dot-... indica il risultato di una generazione multipla

- Il file **generator.py** contiene al suo interno il codice del generatore di grafi scritto interamente in linguaggio Python

- La cartella **reader** contiene al suo interno l'intera struttura relativa al lettore di grafi

- La cartella **result** contiene i risultati delle letture / sperimentazioni su grafi singoli

  1. Al suo interno vi sono H cartelle result-dir-H (il nome viene preso dinamicamente dal grafo in lettura) generate dinamicamente
  2. Ciascuna cartella contiene il singolo risultato di una lettura / sperimentazione su un grafo sotto forma di una coppia di file .init e .out
  3. Ogni gruppo cartella-file.init-file.out rappresenta il risultato di una singola lettura / sperimentazione

- La cartella **m-result** contiene i risultati delle letture / sperimentazioni su molteplici grafi

  1. Al suo interno vi sono W cartelle m-result-dir-W (il nome viene preso dinamicamente dalla cartella relativa alla moltitudine di grafi in lettura) generate dinamicamente
  2. Ciascuna cartella contiene multipli risultati di multiple letture / sperimentazioni su grafo multipli sotto forma di una coppia di file .init e .out
  3. Ogni gruppo cartella-file.init-file.out rappresenta il risultato di una lettura / sperimentazione multipla

- Il file **reader.py** contiene al suo interno il codice del lettore di grafi scritto interamente in linguaggio Python

```
/
├── generator.dir
│   ├── gen.dir
│   │   ├── gen-dir-1.dir
│   │   │   ├── graph-1.edgelist
│   │   │   └── graph-1.dot
│   │   ├── ....
│   │   └── gen-dir-X.dir
│   │       ├── graph-X.edgelist
│   │       └── graph-X.dot
│   ├── m-gen.dir
│   │   ├── m-gen-dir-1.dir
│   │   │   ├── graph-1.edgelist
│   │   │   ├── graph-1.dot
│   │   │   ├── ....
│   │   │   ├── graph-K.edgelist
│   │   │   └── graph-K.dot
│   │   ├── ....
│   │   └── m-gen-dir-Y.dir
│   │       ├── graph-1.edgelist
│   │       ├── graph-1.dot
│   │       ├── ....
│   │       ├── graph-Z.edgelist
│   │       └── graph-Z.dot
│   └── generator.py
└── reader.dir
    ├── result.dir
    │   ├── result-dir-1.dir
    │   │   ├── graph-1.init
    │   │   └── graph-1.out
    │   ├── ....
    │   └── result-dir-H.dir
    │       ├── graph-H.init
    │       └── graph-H.out
    └── m-result.dir
        ├── m-result-dir-1.dir
        │   ├── graph-1.init
        │   └── graph-1.out
        ├── ....
        └── m-result-dir-W.dir
            ├── graph-W.init
            └── graph-W.out
```

3.2 Componenti utilizzati e progettazione

I programmi generator.py e reader.py sono stati interamente scritti utilizzando il linguaggio Python [versione 3.6.5].

Il codice è stato scritto utilizzando differenti editor di testo (vim [neovim], spacemacs, sublime-text, atom,...) e testato su diverse macchine GNU/Linux e Windows, in particolare su differenti shell (bash, zsh, fish) e su cmd.

Per agevolare il processo di progettazione e scrittura del codice l'intera struttura del progetto è stata caricata in una repository sul sito github e gestita in remoto (attraverso il software git).

Per questioni di compatibilità e versatilità è stata utilizzata in modo massiccio la libreria standard relativa al linguaggio Python per effettuare la quasi totalità delle operazioni in entrambi i programmi.

La versione di riferimento della libreria è quella associata alla versione del linguaggio utilizzato, dunque la 3.6.5.

Al fine di trascurare alcuni aspetti relativi alla strutturazione e costruzione dei grafi è stata utilizzata una potente libreria per la creazione e la manipolazione di questi oggetti matematici, ovvero NetworkX.

Tale scelta di progettazione ha permesso al sottoscritto di concentrarsi maggiormente sulla progettazione e sull'ottimizzazione degli algoritmi.

Inoltre tale scelta ha consentito al sottoscritto di rendere totalmente dinamici i processi di creazione e lettura dei grafi, ciò ha facilitato di molto il carico di lavoro in fase di sperimentazione.

La libreria standard del linguaggio Python è stata utilizzata in particolare per rendere totalmente dinamica e cross-platform la gestione del filesystem.

Ciò è stato necessario per garantire il funzionamento asincrono del generatore e del lettore, in modo tale da facilitare il lavoro in fase di sperimentazione.

Le fasi di generazione e lettura dei grafi infatti sono state completamente separate al livello di utilizzo, per fare ciò è stato necessario manipolare efficientemente il filesystem in modo da salvare i grafi creati e i risultati delle sperimentazioni su file.

Tali operazioni sono perfettamente funzionanti sia su sistemi che rispettano lo standard POSIX (Sistemi Unix-like) per il filesystem sia per sistemi che non lo rispettano (Sistemi Windows), dunque l'intero progetto è totalmente cross-platform e può essere facilmente migrato rendendo la portabilità un importante fattore di forza di quest'ultimo.

Il formato principale manipolato dai programmi è il formato .edgelist che analizzeremo in seguito.

In fase di debug è stata utilizzata la libreria di disegno MatPlotLib per rappresentare i grafi, quest'ultima è integrata in modo nativo all'interno della libreria NetworkX, dunque tale scelta di utilizzo ha reso più facili le operazioni di analisi e debug.

Un'altra libreria che è stata utilizzata in fase di analisi e debug per rappresentare i grafi in ambiente GNU/Linux è la libreria di disegno PyGraphViz.

Anche quest'ultima è integrata in modo nativo all'interno della libreria NetworkX.

Il formato di descrizione testuale dei grafi .dot è stato utilizzato solo in fase di debug in ambiente GNU/Linux assieme alla libreria di disegno PyGraphViz, dunque possiamo tralasciare la sua definizione e descrizione poiché non viene utilizzato all'interno dei programmi durante l'esecuzione.

Altre librerie minori sono state scelte in fase di progettazione ed utilizzate all'interno dei programmi, ad esempio la libreria per la colorazione dell'output testuale su terminal emulators cross-platform Colorama è stata utilizzata in fase di analisi e debug per semplificare e rendere più chiara la lettura dell'output relativo all'esecuzione degli algoritmi.

Un altro esempio è l'utilizzo della libreria cross-platform Pick, che rende semplice ed efficace la selezione delle opzioni all'interno dei terminal emulators durante l'esecuzione dei programmi.

L'elenco completo delle librerie utilizzate all'interno dei programmi è il seguente.

- Python Standard Library (Python 3.6.5 - Python 2.7.14)
  https://docs.python.org/3/library/
- NetworkX Library (NetworkX 2.1)
  https://pypi.org/project/networkx/

- Matplotlib Library (Matplotlib 2.2)
  https://pypi.org/project/matplotlib/
- Pick Library (Pick 0.6.4)
  https://pypi.org/project/pick/
- Pydot Library (Pydot 1.2.4)
  https://pypi.org/project/pydot/
- Graphviz Library (Graphviz 0.8.2)
  https://pypi.org/project/graphviz/
- PyParsing Library (PyParsing 2.2.0)
  https://pypi.org/project/pyparsing/
- Colorama Library (Colorama 0.3.9)
  https://pypi.org/project/colorama/

Per il funzionamento completo dei programmi è necessaria l'installazione di Python e dei suddetti componenti attraverso l'uso del modulo pip e/o l'uso di package manager (apt, pacman, ...).

Procediamo ora nella trattazione descrivendo il funzionamento dei programmi generator.py e reader.py.
Saranno poi analizzati in modo approfondito gli algoritmi per il calcolo degli equilibri di Nash e degli ottimi.

3.3 generator.py : il generatore di grafi

Il programma generator.py è un generatore dinamico di grafi scritto interamente in Python in grado di semplificare le operazioni di creazione e manipolazione di questi oggetti matematici e in grado di creare automaticamente tutte le strutture di filesystem necessarie al salvataggio dei grafi generati su file.

Il programma come specificato in precedenza è completamente cross-platform e per ciò che riguarda la gestione del filesystem è stato accuratamente ottimizzato per non generare conflitti e problemi di inconsistenza dei dati.

Principalmente il programma utilizza due moduli principali necessari al corretto funzionamento dello stesso, la Standard Library del linguaggio Python per le funzioni di base e la libreria NetworkX per la creazione e manipolazione dei grafi.

Di seguito vengono riportate le caratteristiche del programma assieme ad un esempio generale di un caso d'uso.

Per prima cosa vengono impostati i motori di disegno, ovvero le librerie MatPlotLib e PyGraphViz.
Quest'ultima è stata utilizzata solo in ambiente GNU/Linux in fase di debug, dunque non è disponibile per l'utente.

In compenso è però disponibile per l'utente la libreria MatPlotLib che consente a quest'ultimo, qualora volesse, di disegnare al termine della generazione i grafi appena creati.
È di fondamentale importanza però specificare che il processo di disegno per grafi di grandi dimensioni è molto dispendioso e dunque può richiedere un tempo considerevole.

Come prima operazione all'avvio del programma è disponibile per l'utente una scelta della classe di grafo da generare effettuata attraverso un interfaccia di selezione da console implementata grazie alla libreria Pick.

Sono disponibili per l'utente 2 modalità di funzionamento per la generazione, la modalità SINGLE MODE (la modalità di default del programma) e la modalità MULTIPLE MODE (accessibile attraverso la selezione dell'opzione MULTIPLE nella scelta della classe).

3.3.1 generator.py : SINGLE MODE

Le classi implementate fanno riferimento a quelle presenti nella libreria NetworkX, ovvero le seguenti (se ne citano solo alcune).

- Classic
- Expanders
- Small
- Random graphs
- Duplication divergence
- ...
- MULTIPLE

Una volta selezionata la classe desiderata l'utente si troverà davanti una nuova interfaccia di selezione da console implementata attraverso l'uso del modulo Pick.

L'opzione MULTIPLE cambia la tipologia di funzionamento del programma, che da SINGLE MODE (modalità di generazione di un singolo grafo per volta) passa a MULTIPLE MODE (nella quale possono essere generati da 1 a n grafi della stessa tipologia in un solo processo di creazione, con n scelto dall'utente).
Concentriamoci ora sulla SINGLE MODE, la default mode del programma.

Questa volta l'utente dovrà scegliere la tipologia di grafo da creare.
Per ogni classe vi sono molteplici tipologie di grafo che l'utente può scegliere di generare.

Le tipologie implementate fanno riferimento a quelle presenti nel modulo NetworkX, in particolare ciascuna tipologia ha un costruttore di libreria corrispondente che provvede a generare le strutture elementari e a comporre l'oggetto matematico richiesto dall'utente.

Solo per far comprendere al lettore la vastità delle scelte possibili per l'utente in fase di selezione del grafo da generare, vengono qui si seguito riportate le tipologie di grafi implementate per la sola classe di grafi Classic.

- balanced tree
- complete graph
- circular ladder graph
- cycle graph
- dorogovtsev goltsev mendes graph
- ladder graph
- lollipop graph
- path graph
- star graph
- turan graph
- wheel graph
- ...

Una volta selezionata la tipologia di grafo da implementare, l'utente potrà inserire i parametri di creazione per quest'ultimo.

I parametri associati ai grafi variano da tipologia a tipologia e sono necessari e fondamentali per la corretta generazione del grafo scelto.

Per una migliore comprensione, ad esempio i parametri associati alla tipologia balanced tree sono il branching factor e l'height dell'albero, invece per la tipologia di grafo complete graph vi è un unico parametro da passare al programma, il node number del grafo.
L'esempio ovviamente ricopre solo 2 tipologie di grafo, ma ciò vale per ogni tipologia implementata nel programma.

A questo punto l'utente è chiamato a scegliere quale tipologia di dato utilizzare per codificare i pesi associati agli archi del grafo, peso flottante (tipo float) o peso intero (tipo int).
Possiamo tralasciare il tipo flottante, dato che quest'ultimo, non essendo interessante e significativo per la sperimentazione, è stato eliminato in favore del tipo intero.

L'utente dunque dovrà scegliere il range di valori all'interno dei quali oscilleranno randomicamente i pesi interi associati agli archi del grafo.
È possibile scegliere solo pesi interi positivi, ovvero i valori minimo e massimo del range devono essere compresi tra $0 \geq minimo \geq massimo \geq n$, con $n \to \infty$.

Parte IV

Sperimentazione