Algorithmic Game Theory

Elisa Gioacchini - Daniele Pasquini - Giacomo Scornavacca

13/12/2013

Indice

Capitolo 1

Licenza

Capitolo 2

Introduzione

L'Algorithmic Game Theory è l'area di intersezione tra la Game Theory e la Theory of Algorithms il cui obiettivo è l'analisi e la progettazione di algoritmi in ambienti strategici.

Game Theory (Teoria dei Giochi): scienza matematica che studia le situazioni di conflitto ricercandone soluzioni competitive e cooperative tramite modelli. Si tratta dunque dell'analisi delle decisioni individuali in situazioni di interazione con altri soggetti rivali (due o più), tali per cui le decisioni di uno possono influire sui risultati conseguibili dall'altro/i secondo un meccanismo di retroazione, e finalizzate al massimo guadagno del soggetto. In parole povere la Game Theory si sforza di predire l'esito di un gioco (soluzione) prendendo in considerazione il comportamento individuale del giocatore, fa uso di strumenti e modelli che descrivono problemi computazionali su agenti razionali.

Theory of Algorithms (Teoria degli Algoritmi e della Complessità): scienza informatica che si occupa dell'analisi e della progettazione degli algoritmi. Pone una particolare enfasi sulla complessità computazionale degli algoritmi, intesa come utilizzo delle risorse computazionali (memoria occupata, tempo di calcolo, eccetera) necessarie ad eseguirli.

L'**Algorithmic Game Theory** è un campo di studio recente, legato in parte ai sistemi distribuiti strategici (quindi non-cooperativi come ad esempio i giochi su rete), presenta e integra aspetti provenienti da entrambe le discipline. Semplificando, l'Agorithmic Game Theory da una parte usa gli strumenti della Teoria dei Giochi per analizzare scenari e problematiche tipiche dell'area informatica (per esempio protocolli su reti utilizzati da agenti egoistici) e dall'altra complementa i risultati tipici della Teoria dei Giochi valutandone gli aspetti legati alla complessità computazionale (per esempio valutando non solo l'esistenza di un certo equilibrio, ma anche la possibilità di calcolarlo in tempo ragionevole).

2.1 Game Theory

Un gioco è ben definito una volta che si sono definite le seguenti cose:

- n, il numero di giocatori che prendono parte al gioco;
- **un insieme di regole**, descrivono cosa i giocatori possono fare, ogni player i può scegliere la propria *strategia* tra un numero finito di elementi $S_i \in \{s_1, s_2, \ldots, s_k\}$, si definisce inoltre quando i giocatori possono scegliere o modificare la propria strategia;
- **payoff**, il guadagno del singolo giocatore relativo alla configurazione di strategie di tutti gli n giocatori;
- **soluzioni**, un insieme di outcomes.

Lo **stato globale del sistema** è determinato dal vettore delle scelte di strategie degli n player $[S_1, S_2, \ldots, S_n]$. A volte può essere utile analizzare lo stato del sistema dal punto di vista di uno specifico player i, per il quale utilizzeremo una notazione in cui ci si limita a separare le strategie degli altri players dalla propria $S_{-i} = [S_1, \ldots, S_{i-1}, S_{i+1}, \ldots, S_n]$.

L'**obiettivo** di ogni player è massimizzare una funzione locale che indica il payoff $\mho$, che è determinata dalla propria strategia e dallo stato globale del sistema (quindi dalle scelte degli altri players). Il **costo sociale** dello stato globale del sistema viene misurato attraverso un'altra funzione, spesso definita come la somma dei payoff dei singoli player.

Un **equilibrio** è una particolare configurazione di strategie nella quale ciascuno dei giocatori non ha interesse nel modificare la propria strategia, l'equilibrio non è necessariamente unico.

Queste poche definizioni sono sufficienti per affrontare i primi problemi di Game Theory.

2.2 Il dilemma del prigioniero

Descrizione: due malfattori commettono insieme due crimini, uno di natura lieve e uno grave; entrambi vengono arrestati e rinchiusi in celle separate senza avere l'opportunità di comunicare. I carcerieri possiedono prove solo per quanto riguarda il crimine lieve e cercano di indurli a collaborare confessando l'implicazione dell'altro malfattore nel crimine grave. La confessione implica uno sconto di pena per chi collabora. Questa è la condanna prevista per ciascun crimine:

- 2 anni di carcere per un crimine lieve;
- 5 anni di carcere per un crimine lieve e un crimine grave;
- sconto di pena di 1 anno per una confessione.

Per esaminare i casi possibili si introduce la **matrice dei costi**, uno strumento utile a valutare il set di strategie dei giocatori.

In questo caso questa è la matrice dei costi:

	Non confessa	**Confessa**
Non confessa	$(2, 2)$	$(5, 1)$
Confessa	$(1, 5)$	$(4, 4)$

Tabella 2.1: Matrice di costi: il primo valore si riferisce al payoff del player 1, il secondo al payoff del player 2.

Poniamoci nei panni di uno dei prigionieri nel tentativo di ottimizzare il payoff (minimizzare il numero degli anni di carcere) e analizziamo le riflessioni del singolo malfattore con l'ausilio della matrice dei costi. Facendo una analisi per casi:

- se l'altro prigioniero sceglie di non confessare, il malfattore può scegliere tra confessare che fornisce un payoff di 1 oppure non confessare, con un payoff di 2;
- il caso in cui l'altro prigioniero sceglie di confessare, il payoff conseguente al confessare sarà minore che nel caso in cui si confessa (4 contro 5).

Quindi essendo il malfattore egoista e razionale, in entrambe le situazioni avrebbe interesse a confessare, a prescindere dalla strategia dell'avversario. Il ragionamento è valido anche per l'altro prigioniero, quindi il gioco si attesta su una vettore di strategie (Confessa, Confessa) che viene chiamato Equilibrio a Strategia Dominante (Dominant Strategy Equilibrium).

Naturalmente se viene esaminato il costo sociale del vettore delle scelte (Non confessa, Non confessa) esso è nettamente inferiore a quello del vettore (Confessa, Confessa) per un valore di 4 contro un valore di 8 ma non è un equilibrio pertanto i malfattori non sceglieranno la prima configurazione nel set delle loro strategie.

2.3 Internet Service Provider (ISP) routing game

Descrizione: La rete in Figura **??** è gestita da due player. Il player 1, che si fa carico della gestione e quindi dei costi relativi agli archi rossi, e il player 2, che si fa carico dei costi degli archi blu. Gli archi neri, invece, rappresentano link di comunicazione per i quali l'utilizzo non comporta nessun costo. Il player i ($i = 1, 2$) vuole mandare un messaggio dal nodo S_i al nodo T_i e può scegliere due cammini sulla rete: uno che passa per il nodo C e l'altro che passa per il nodo S. Ogni player è strategico e vuole spedire il messaggio in modo da sostenere un costo il più basso possibile. Si noti, inoltre, che tale costo dipende non solo dal cammino che sceglie di usare per spedire il proprio messaggio ma anche dal cammino scelto dall'altro giocatore, poiché il player dovrà farsi carico del costo di attraversamento dei propri archi qualora il messaggio dell'altro giocatore li usasse. Cosa ci aspettiamo faranno i due giocatori? Modelliamo la situazione come un gioco. Di seguito è riportata la matrice dei costi. Si noti che tale matrice è identica a quella del Dilemma dei Prigionieri. I due giocatori,

quindi, sceglieranno entrambi di spedire il proprio messaggio verso il nodo C (e avranno un costo entrambi di 4). Da un punto di vista sociale, invece, la soluzione migliore per entrambi sarebbe spedire i messaggio verso il nodo S.

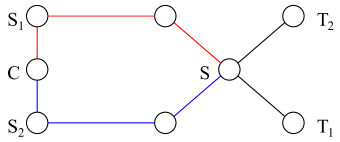


Figura 2.1: Costo degli archi

	Verso S	**Verso C**
Verso S	$(2, 2)$	$(5, 1)$
Verso C	$(1, 5)$	$(4, 4)$

Tabella 2.2: Matrice di costi: il primo valore si riferisce al payoff (costo) del player 1, il secondo al payoff (costo) del player 2.

2.4 Equilibri in Strategie Dominanti

I due giochi visti fino ad adesso sono entrambi giochi che ammettono delle soluzioni molto forti, dette equilibri in strategie dominanti. Intuitivamente sono giochi in cui ogni giocatore ha una strategia che è la migliore possibile indipendentemente da quello che possono giocare gli altri giocatori. Più formalmente, un *equilibrio in strategie dominanti* è un profilo di strategie $(S_1^*, S_2^*, \ldots, S_n^*)$ tale che ogni S_i^* è una *strategia dominante* per il player i, ovvero, una strategia S_i^* per cui vale che per ogni altro possibile profilo di strategie S_{-i} giocato dagli altri player e per ogni strategia S_i giocabile dal player i vale:

$$p_i(S_{-i}, S_i^*) \geq p_i(S_{-i}, S_i)$$

dove p_i è la funzione payoff del player i. Quindi scegliere la strategia S_i^* assicura il miglior payoff.

Si noti che la definizione è data nel caso in cui la funzione payoff del player è una funzione da massimizzare (tali giochi sono spesso detti giochi in forma di massimizzazione). Una definizione simile per i giochi in forma di minimizzazione (in cui la funzione payoff è di fatto una funzione costo che il giocatore vuole minimizzare) si ottiene scambiando la relazione $\geq$ con la relazione $\leq$.

Questo concetto di equilibrio è molto forte e ha una forte valenza predittiva: se un gioco ammette un equilibrio in strategie dominanti, poiché un giocatore non necessita di alcuna conoscenza sulle scelte delle strategie degli altri giocatori e ha sempre convenienza a giocare la sua strategia dominante, è ragionevole

aspettarsi che l'outcome del gioco sarà proprio l'equilibrio in strategie dominanti. C'è da dire, però, che pochi giochi ammettono un equilibrio di questo tipo.

2.5 Equilibrio di Nash

Un concetto di equilibrio più debole di quello in strategie dominanti è quello di equilibrio di Nash (in strategie pure). Intuitivamente, questo tipo di equilibrio è un profilo di strategie (e quindi uno stato del sistema) in cui a nessun giocatore conviene cambiare strategia unilateralmente. Concentrandoci su un gioco in forma di massimizzazione, più formalmente un *equilibrio di Nash* (in strategie pure) è un profilo di strategie S^* in cui, per ogni giocatore i, e per ogni possibile sua strategia S_i, vale

$$p_i(S^*) \geq p_i(S^*_{-i}, S_i).$$

Nel caso in cui il gioco fosse in forma di minimizzazione (e quindi la funzione payoff è di fatto una funzione costo), ancora una volta è sufficiente sostituire la relazione di $\geq$ con quella di $\leq$. Si noti inoltre che, se si considera una scenario di gioco ripetuto, in cui, a partire da un certo profilo di strategie iniziali, si da la possibilità a ogni giocatore di cambiare a turno la propria strategia (qualora ne avesse una migliorativa), se questa dinamica converge su un certo profilo di strategie, questo deve necessariamente essere un equilibrio di Nash. In molti casi è possibile pensare a un tale profilo come ad uno stato del sistema che una volta raggiunto è uno stato stabile, ovvero non cambierà più (perché i giocatori non hanno unilateralmente interesse ad abbandonarlo).

2.6 Sulla bontà degli equilibri: Prezzo dell'Anarchia e Prezzo della Stabilità

Come già accennato, un profilo di strategie può essere pensato come uno stato del sistema, un risultato dell'interazione fra i giocatori. A un generico stato o outcome è possibile associare un valore che ne descrive la qualità in termini di qualche funzione obiettivo. Tale funzione obiettivo è generalmente chiamata funzione di scelta sociale o funzione sociale. Una funzione sociale particolarmente significativa è quella definita come somma dei payoff dei giocatori. Si noti che questa funzione, in caso di problema di massimizzazione, misura l'utilità complessiva dei player e in caso di gioco di minimizzazione misura il costo complessivo di tutti i giocatori, e quindi è in ogni caso una misura della qualità globale dello stato del gioco. Per esempio, il profilo di strategie che ottimizza la funzione sociale è in un certo sento l'outcome più desiderabile da un punto di vista globale, ma non è detto che tale profilo sia un equilibrio e quindi non ci si può aspettare che il sistema mantenga tale stato in presenza di giocatori egoistici.

Proprio per queste ragioni, i profili a cui si è interessati sono quelli in equilibrio (nel nostro caso stiamo parlando ora di equilibri di Nash) ed è interessante capire, sotto l'assunzione che il gioco possa stabilizzarsi su un certo equilibrio, quanto tale equilibrio può essere socialmente desiderabile. A questo scopo si introducono due misure che hanno l'obiettivo di misurare quanto la qualità di

un equilibrio possa degradare rispetto a una soluzione (stato) socialmente ottimo. Queste misure sono il *Prezzo dell'Anarchia (PoA)* e il *Prezzo della Stabilità (PoS)*.

Sia G un gioco, $f(s)$ una funzione sociale di minimizzazione (rispettivamenete, di massimizzazione). Sia S l'insieme di equilibri di Nash del gioco e OPT il profilo di strategie che ottimizza f. Si definisce Prezzo dell'Anarchia per un gioco in forma di minimizzazione (rispettivamente massimizzazione) il seguente rapporto:

$$PoA_G = \sup_{s \in S} \frac{f(s)}{f(OPT)} \quad \left(\text{ rispettivamente } \inf_{s \in S} \frac{f(s)}{f(OPT)} \right).$$

Il Prezzo della Stabilità è invece è definito nel seguente modo:

$$PoS_G = \inf_{s \in S} \frac{f(s)}{f(OPT)} \quad \left(\text{ rispettivamente } \sup_{s \in S} \frac{f(s)}{f(OPT)} \right).$$

Intuitivamente, il PoA misura la qualità del peggiore equilibrio mentre il PoS quella del miglior equilibrio. Un gioco che ha un PoA basso è un gioco in cui tutti gli equilibri non sono molto lontani dall'ottimo sociale e mentre giochi con un PoA alto sono giochi in cui la mancata cooperazione fra i giocatori può portare ad outcome che sono molto lontani in termini di qualità dall'ottimo sociale. Il PoS di un gioco invece intuitivamente fornisce una misura di quanto è necessario pagare per avere un outcome stabile. Giochi con PoS alto sono giochi in cui ogni equilibrio è socialmente poco desiderabile, mentre giochi con PoS basso sono giochi per cui c'è almeno un equilibrio che non si discosta molto dall'ottimo sociale.

Capitolo 3

Network Formation Game

Le categorie di giochi che rientrano nei Network Formation Game (NFG) consentono di modellare il processo distribuito e non cooperativo di interazione di player su reti. Se la finalità dei player è lo sfruttamento di una rete esistente o la sua costruzione parleremo di due tipologie di gioco:

- Global Connection Game
- Local Connection Game

Gli obiettivi comuni sono i seguenti:

- Minimizzare i costi di costruzione
- Ottenere una rete di qualità, che offra un buon servizio.

Gli NFG possono essere usati per modellare:

- Reti sociali, in cui ogni arco rappresenta una relazione
- Reti di telecomunicazioni
- Reti P2P
- Sottoreti

In questi contesti ci concentreremo sull'analisi di reti stabili, ovvero di reti costituite da equilibri di Nash, con l'obiettivo di stimare i costi in termini di perdita di efficienza rispetto a soluzioni centralizzate. Per valutare la qualità della rete verrà considerata la somma dei costi sostenuti dai giocatori.

3.1 Global Connection Game

3.1.1 Definizione

Sia $G(V, E)$ un grafo diretto e pesato, con pesi c_e non negativi, dove c_e rappresenta il costo di acquisto dell'arco e. Partecipano al gioco k player, ciascuno di essi rappresentato dalla coppia di nodi $i = (s_i, t_i)$, dove s_i è il nodo sorgente e t_i il nodo target. L'obiettivo del player i è collegare con un cammino diretto P_i s_i a t_i, spendendo il meno possibile. La strategia di un player i è un cammino

p_i da s_i a t_i.
Dato un vettore (o profilo) di strategie $P = [p_1, \ldots, p_k]$, un sottoinsieme E' degli archi di partenza,

$$E'(S) = \bigcup_i p_i$$

determinando un grafo $G' = (V, E')$.

Il costo del singolo player i viene ottenuto ripartendo il costo della rete tra i giocatori (Fair o Meccanismo di condivisione dei costi di Shapley). Per la precisione:

$$cost_i(P) = \sum_{e \in p_i} \frac{c_e}{k_e(P) \equiv k_e}$$

dove k_e rappresenta il numero di player che hanno scelto l'arco e per il proprio cammino

Richiami

- Dato un vettore di strategie P, $G(P)$ (grafo indotto dalle strategie in P) è stabile se P è un equilibrio di Nash.
- La tipologia di giochi che siamo in grado di analizzare alla ricerca di equilibri di Nash sono quelli iterativi: dato un profilo di strategie iniziale, i player a turno scelgono la propria nuova strategia conoscendo quelle degli altri.

Il costo sociale della rete è dato dalla somma dei costi dei singoli player, o equivalentemente dalla somma dei costi degli archi della rete G' (poichè $\sum_{i \in [1,k]} \frac{c_e}{k_e} = c_e$).

$$cost(G') = \sum_{i \in [1,k]} \sum_{e \in p_i} \frac{c_e}{k_e} = \sum_{e \in E'} c_e$$

3.1.2 Esempi

Esempio 1 Sia G(V, E) un grafo diretto e pesato, come mostrato in figura **??**. Supponiamo di avere due player, 1 e 2. Il player 1 deve raggiungere t_1 da s_1, il player 2 deve raggiungere t_2 da s_2.

Supponiamo che entrambi i player scelgano lo stesso cammino (figura **??**). E' possibile notare che i costi sono pari a:

$$cost_1 = 7,\ cost_2 = 6,\ sociale = 13$$

Dove il costo sociale è dato dalla somma dei costi degli archi che costituiscono la soluzione. Una rete è quindi ottima se è il sottografo G' scelto tra tutti i sottografi di G che contengono un cammino da s_i a t_i $\forall i$, di costo minimo.

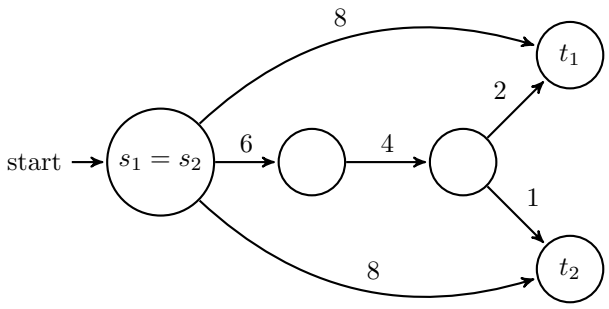


Figura 3.1: G(V, E)

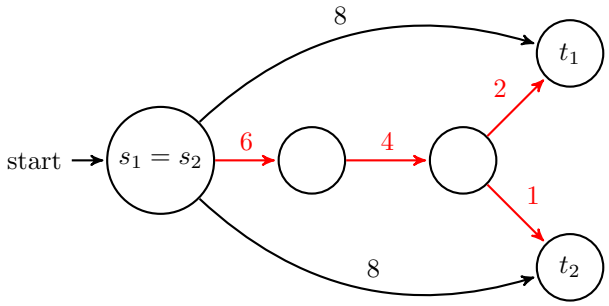


Figura 3.2: N(S)

Esempio 2 Sia $G(V, E)$ il grafo diretto e pesato mostrato in figura **??**. Quale è in questo caso la rete ottima?

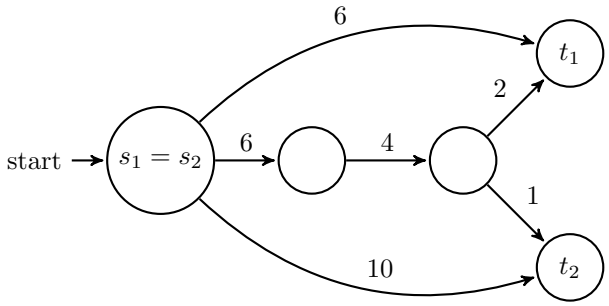


Figura 3.3: G(V, E)

E' evidente che la rete ottima è la rete mostrata nel grafo in figura **??** poiché il costo dell'ottimo sociale è pari a 13.

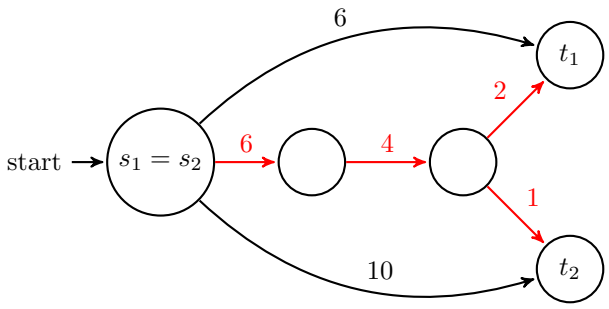


Figura 3.4: N(S)

Attenzione: nonostante sia ottima non è una rete stabile. Infatti il player 1, nel turno successivo, sceglierebbe l'arco (s_1, t_1) (figura **??**) dal momento che $cost_1 = 6$, che è minore del $cost_1 = 7$ della rete ottima.

totale = 17

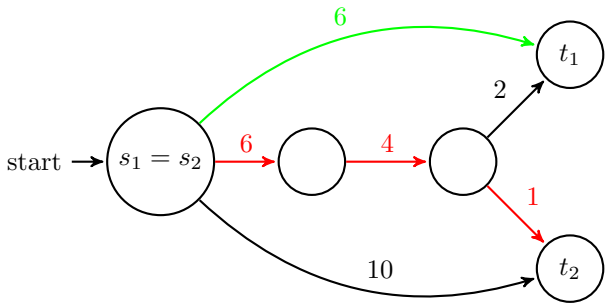


Figura 3.5: N(S)

La rete, però, ancora non si può definire stabile. Con il cambio di strategia del player 1, anche al player 2 conviene cambiare strategia. Infatti utilizzando gli archi attuali non pagherebbe $cost_2 = 6$ ma $cost_2 = 11$. Per questo il player 2 sceglierebbe l'arco (s_2, t_2) tale che $cost_2 = 10$ (figura **??**).

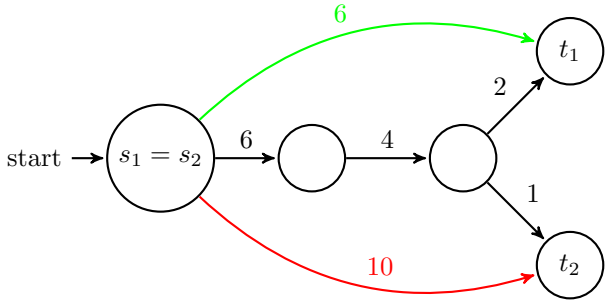


Figura 3.6: N(S)

La rete a questo punto è stabile perchè formata da un equilibrio di Nash.

$cost_1 = 6$, $cost_2 = 10$, $totale = 16$.

Esempio 3 Sia $G(V, E)$ un grafo diretto e pesato (figura **??**). La rete ottima è rappresentata dal grafo in figura **??**.

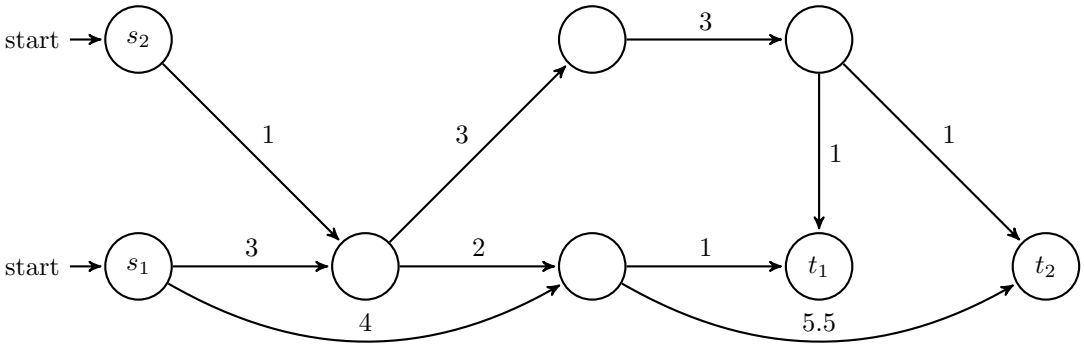


Figura 3.7: N(S)

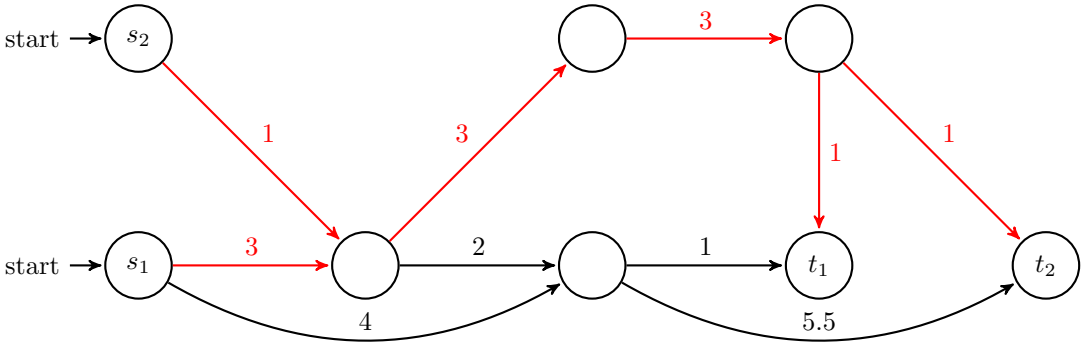


Figura 3.8: N(S)

$cost_1 = 7$, $cost_2 = 5$, $totale = 12$.

Ma non è una rete stabile, poiché seguendo la dinamica della best response, il player 1 può diminuire il suo costo (figura **??**). In questa nuova situazione invece la rete è stabile e i costi sono pari a:

$cost_1 = 5$, $cost_2 = 8$, $totale = 13$.

Nonostante questo sia un equilibrio di Nash, ne esiste un secondo migliore, come si evince dalla figura **??**:

$cost_1 = 5$, $cost_2 = 7,5$, $totale = 12,5$.

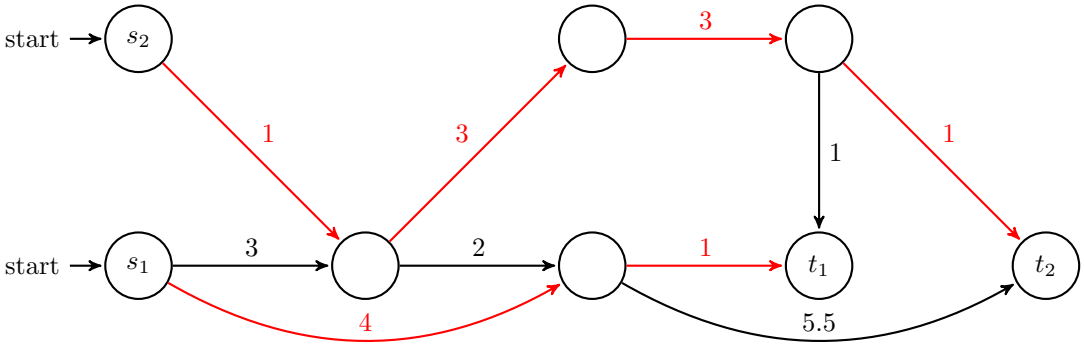


Figura 3.9: N(S)

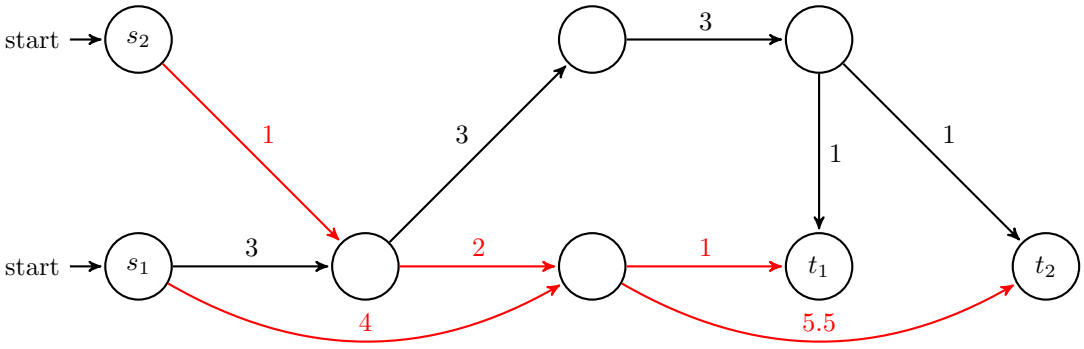


Figura 3.10: N(S)

3.1.3 Analisi del gioco

- Esiste sempre una rete stabile, ovvero un equilibrio di Nash?
- E' possibile stimare il prezzo dell'anarchia PoA?
- E' possibile stimare il prezzo della stabilità PoS?
- A partire da qualsiasi configurazione, un gioco ripetuto converge sempre ad una rete stabile, e quindi ad un equilibrio di Nash?

Si vedrà in particolare che, se ogni player i gioca secondo una strategia better response, allora il gioco converge ad un equilibrio di Nash.

Richiami

Data una rete G, definiamo:

- PoA del gioco in G = $\max_{S \in \text{NE}} \frac{cost(S)}{cost(S^*_G)}$
- PoS del gioco in G = $\min_{S \in \text{NE}} \frac{cost(S)}{cost(S^*_G)}$

L'obiettivo è stimare PoA e PoS nel caso peggiore, poiché ci interessa osservare come cambia il valore del rapporto al variare di G.

- PoA = $\max_G$ PoA in G
- PoS = $\min_G$ PoS in G

Formalizzazione

- $x = (x_1, x_2, x_3, ..., x_k) \rightarrow$ Profilo di strategie dei k player.
- $x_{-i} = (x_1, x_2, ..., x_{i-1}, x_{i+1}, ..., x_k) \rightarrow$ Il player i ha cambiato strategia, il vettore viene decurtato della componente i-esima.
- $x = (x_{i-1}, x_i)$
- Sia G una rete/grafo diretto e pesato, il costo o la lunghezza di un percorso π in G da un nodo u a un nodo v è: $\sum_{e \in \pi} c_e$
- $d_g(u, v)$: distanza in G dal nodo u al nodo v, ovvero la lunghezza del cammino minimo in G da u a v.

3.1.4 PoA: un Lower Bound

Sia $G(V, E)$ un grafo diretto e pesato (figura **??**):

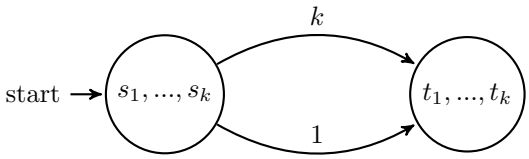


Figura 3.11: G(V, E)

I due archi del grafo rappresentano le due possibili strade percorribili dai giocatori: se k_e player utilizzano l'arco inferiore il costo del singolo sarà $1/k_e$ e il costo totale dell'arco sarà 1; quello superiore ha costo variabile in funzione del numero di player k_e che lo utilizzano, in questo caso il costo del singolo sarà 1 e il costo dell'arco sarà k_e.

La rete sociale ottima ha costo pari a 1, nella soluzione in cui ogni player sceglie di percorrere l'arco inferiore. Il miglior equilibrio di Nash si ha quando la configurazione iniziale prevede che tutti i player usino l'arco inferiore (il costo per ogni player è pari a $1/k$), e in questo caso PoS in G = 1, mentre il peggior equilibrio di Nash si ha quando la configurazione iniziale prevede che tutti i player usino l'arco superiore (il costo per ogni player è pari a $k/k = 1$) e a nessuno, quindi, converrebbe spostarsi sull'arco inferiore. In questo caso PoA in G = $k/1 = k$.
Quindi: PoA del gioco è $\leq k$

Teorema 1. *Il PoA in un Global Connection Game con k player è al più k.*

Dimostrazione. Sia S un equilibrio di Nash qualsiasi e sia S^* una strategia socialmente ottima. Consideriamo un player i: tra tutti i possibili cammini che può scegliere vi è anche il cammino minimo. Indichiamo con π_i il cammino minimo in G da s_i a t_i. Si ha:

$$cost_i(S) \leq cost_i(S_{-i}, \pi_i) \leq d_G(s_i, t_i) \leq cost(S^*)$$

Dove:

- $cost_i(S)$ costo dell'equilibrio di Nash per il giocatore i;
- $cost_i(S_{-i}, \pi_i)$ costo della strategia cammino minimo per il giocatore i (è un upper bound per la definizione di equilibrio di Nash);
- $d_G(s_i, t_i)$ costo minimo della soluzione per il player i, dove ogni arco della soluzione viene comprato solo dal player i;
- $cost(S^*)$ costo della soluzione socialmente ottima, somma di tutti i costi delle Strategie dei giocatori. È upper bound del costo minimo della soluzione di un qualsiasi player.

Quindi:

$$cost(S) = \sum_i cost_i(S) \leq kcost(S^*) \tag{3.1}$$

□

3.1.5 PoS: un Lower Bound

Sia $G(V, E)$ il grafo in figura **??**.

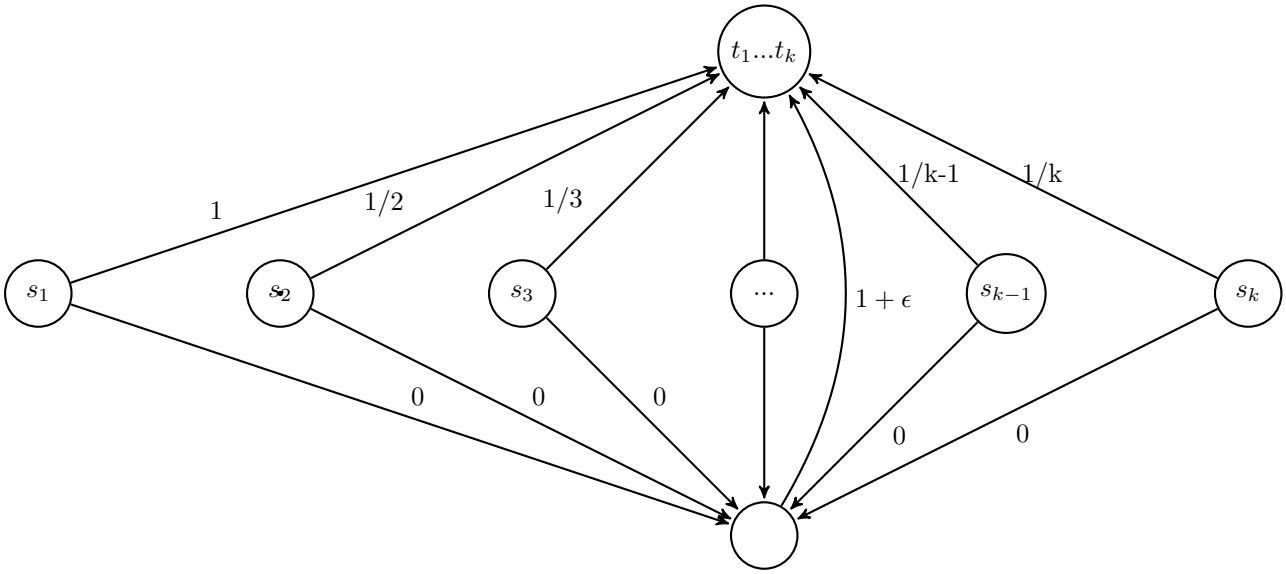


Figura 3.12: G(V, E)

L'ottimo sociale è pari a $1 + \epsilon$ nella soluzione in cui tutti i player pagano $(1 + \epsilon)/k$ passando dal nodo inferiore e dividendo il costo dell'arco uscente da esso. La soluzione però non è un equilibrio di Nash poiché al player s_k conviene cambiare strategia passando per l'arco dal costo $1/k < (1 + \epsilon)/k$. Si noti che, inevitabilmente, con il cambio di strategia di s_k, anche a s_{k-1} conviene cambiare strategia. E a seguire gli altri player cambieranno strategia, finché non si arriva ad un equilibrio di Nash, nel momento in cui tutti passano dagli archi diretti verso il nodo terminale.

$$\begin{aligned} \text{costo sociale NE } &= \sum_{j=1}^{k} \frac{1}{j} = H_k \leq ln(k) + 1 \\ PoS &= \frac{ln(k) + 1}{1 + \epsilon} = H_k \end{aligned} \tag{3.2}$$

Ovvero:

$$PoS \geq H_k$$

Dove H_k è il k-esimo numero armonico.

3.1.6 Funzione potenziale

Definizione Per ogni gioco finito, una funzione potenziale Φ è una funzione che mappa ogni profilo/vettore di strategie S ad un qualche numero $\in \mathbb{R}$ e $\forall S = (S_1, ..., S_k)$, se il player i cambia strategia da S_i a S'_i con $S'_i \neq S_i$ allora $S' = (S_{-i}, S'_i)$ e:

$$\Phi(S) - \Phi(S') = cost_i(S) - cost_i(S') \tag{3.3}$$

Se un gioco ammette una funzione potenziale, allora si definisce gioco potenziale. Questa proprietà risulta essere molto utile per stimare un gioco che la possiede.

Teorema 2. *Ogni gioco potenziale ha almeno un equilibrio di Nash, che coincide con il vettore di strategie S che minimizza* $\Phi(S)$.

Dimostrazione. Supponiamo che, a partire da S che minimizza $\Phi(S)$, un player i cambi strategia, dando vita ad un nuovo vettore di strategie S'. Si ha che:

$$\begin{aligned} \Phi(S) - \Phi(S') &= cost_i(S) - cost_i(S') \\ \text{Poiché } \Phi(S) - \Phi(S') \leq 0 &\Rightarrow cost_i(S) - cost_i(S') \leq 0 \\ \text{Quindi: } cost_i(S) &\leq cost_i(S') \end{aligned} \tag{3.4}$$

Ovvero nessun giocatore ha interesse a modificare la propria strategia poiché non è in grado di migliorare il proprio payoff, quindi S è un equilibrio di Nash. □

Teorema 3. *ogni gioco potenziale finito ha almeno un Equilibrio di Nash, il vettore di strategie S associato all'equilibrio minimizza* $\Phi(S)$.

Dimostrazione. la dinamica di Better Response simula la ricerca di un minimo per la funzione Φ, infatti ogni cambio di strategia diminuisce strettamente il valore di Φ e la soluzioni sono finite. □

Nota Nel Global Connection Game se un player decide di cambiare strategia, può scegliere la best response in base ai cammini degli altri player. La ricerca della best response è calcolabile in tempo polinomiale tramite il cammino minimo.

Teorema 4. *Supponiamo di avere un gioco potenziale con una funzione potenziale* $\Phi(S)$*, e assumiamo che per ogni outcome S si ha:*

$$cost(S)/A \leq \Phi(S) \leq Bcost(S) \tag{3.5}$$

Per qualche $A, B > 0$*. Allora il PoS è al più AB.*

Dimostrazione. Si prendono due vettori di strategie:

- S' è il vettore di strategie che minimizza Φ (è un equilibrio di Nash).
- S^* è il vettore di strategie che minimizza il costo sociale.

Si ha che:

$$cost(S')/A \leq \Phi(S') \leq \Phi(S^*) \leq Bcost(S^*) \tag{3.6}$$

Quindi:

$$cost(S') \leq ABcost(S^*)$$

□

3.1.7 Metodo della funzione potenziale e GCG

Sia Φ la funzione potenziale che manda ogni vettore di strategie S in un valore in $\mathbb{R}$ e sia H_k il numero armonico:

$$\begin{gathered} \Phi(S) = \sum_{e \in E} \Phi_e(S) \\ \text{dove } \Phi_e(S) = c_e H_{k_e(S)} \\ \text{e } H_0 = 0 \end{gathered} \tag{3.7}$$

Lemma 1. *Sia $S = (P_1, ..., P_k)$ e sia P_i' un percorso alternativo per il player i, definiamo quindi un nuovo vettore di strategie $S' = (S_{-i}, P_i')$. Allora Φ è una funzione potenziale per il GCG quindi vale:*

$$\Phi(S) - \Phi(S') = cost_i(S) - cost_i(S') \tag{3.8}$$

Dimostrazione. Voglio mostrare che $\Phi(S) - \Phi(S') = cost_i(S) - cost_i(S')$ dove $cost_i(S) - cost_i(S') = \Delta(c)$ e $\Phi(S) = \sum_{e \in E} c_e H_{k_e(S)}$.

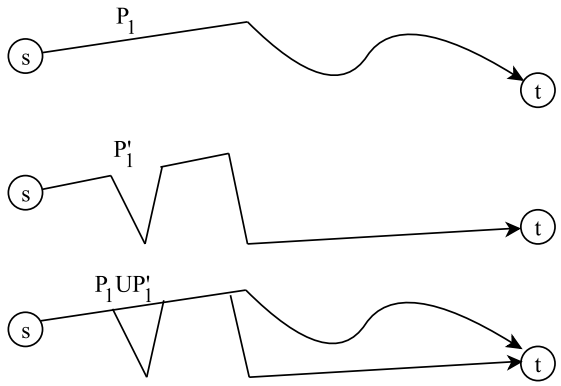


Figura 3.13: Cambio di strategia

$$\Phi(S) - \Phi(S') = \sum_{e \in E} c_e H_{k_e(S)} - \sum_{e \in E} c_e H_{k_e(S')}$$

Avendo definito $H_0 = 0$, gli archi che non vengono percorsi hanno costo nullo, in più gli archi che non subiscono variazioni dal cambiamento di scelta del giocatore i hanno stesso valore nelle due sommatorie, quindi ha senso considerare solo gli archi di P_i e P_i':

$$\sum_{e \in E} c_e H_{k_e(S)} - \sum_{e \in E} c_e H_{k_e(S')} = \sum_{e \in P_i \cup P_i'} c_e H_{k_e(S)} - \sum_{e \in P_i \cup P_i'} c_e H_{k_e(S')}$$

L'insieme unione è esprimibile come unione disgiunta dei seguenti insiemi:

$$P_i \cup P_i' = (P_i \cap P_i') \cup (P_i \setminus P_i') \cup (P_i' \setminus P_i)$$

Quindi le sommatorie diventano:

$$\sum_{e \in P_i \cap P_i'} c_e H_{k_e(S)} - \sum_{e \in P_i' \cap P_i} c_e H_{k_e(S')} +$$
$$+ \sum_{e \in P_i \setminus P_i'} c_e H_{k_e(S)} - \sum_{e \in P_i \setminus P_i'} c_e H_{k_e(S')} +$$
$$+ \sum_{e \in P_i' \setminus P_i} c_e H_{k_e(S)} - \sum_{e \in P_i' \setminus P_i} c_e H_{k_e(S')}$$

Per le proprietà del modello so che:

$$\begin{aligned} &e \in P_i \cap P_i' \Rightarrow k_e(S) = k_e(S') \\ &e \in P_i \setminus P_i' \Rightarrow k_e(S') = k_e(S) - 1 \\ &e \in P_i' \setminus P_i \Rightarrow k_e(S') = k_e(S) + 1 \end{aligned} \tag{3.9}$$

Quindi le prime due sommatorie hanno lo stesso valore e si annullano, restano da stabilire le relazioni tra le altre:

$$\sum_{e \in P_i \setminus P_i'} c_e (H_{k_e(S)} - H_{k_e(S')}) + \sum_{e \in P_i' \setminus P_i} c_e (H_{k_e(S)} - H_{k_e(S')}) =$$

$$= \sum_{e \in P_i \setminus P_i'} c_e (H_{k_e(S)} - H_{k_e(S)-1}) + \sum_{e \in P_i' \setminus P_i} c_e (H_{k_e(S')-1} - H_{k_e(S')}) =$$

$$= \sum_{e \in P_i \setminus P_i'} c_e \left(\frac{1}{k_e(S)} \right) - \sum_{e \in P_i' \setminus P_i} c_e \left(\frac{1}{k_e(S')} \right)$$

Come già visto gli insiemi P_i e P_i' sono entrambi esprimibili con due unioni disgiunte:

- $P_i = (P_i \cap P_i') \cup (P_i \setminus P_i')$;
- $P_i' = (P_i' \cap P_i) \cup (P_i' \setminus P_i)$.

E inoltre:

$$\sum_{e \in P_i \cap P_i'} c_e \left(\frac{1}{k_e(S)} \right) - \sum_{e \in P_i' \cap P_i} c_e \left(\frac{1}{k_e(S')} \right) = 0$$

Quindi vale:

$$\sum_{e \in P_i \setminus P_i'} c_e \left(\frac{1}{k_e(S)} \right) - \sum_{e \in P_i' \setminus P_i} c_e \left(\frac{1}{k_e(S')} \right) =$$

$$= \sum_{e \in P_i \setminus P_i'} c_e \left(\frac{1}{k_e(S)}\right) + \sum_{e \in P_i \cap P_i'} c_e \left(\frac{1}{k_e(S)}\right) - \left[\sum_{e \in P_i' \setminus P_i} c_e \left(\frac{1}{k_e(S')}\right) + \sum_{e \in P_i' \setminus P_i} c_e \left(\frac{1}{k_e(S')}\right)\right] =$$

$$= \sum_{e \in P_i} \frac{c_e}{k_e(S)} - \sum_{e \in P_i'} \frac{c_e}{k_e(S')} =$$

$$= cost_i(S) - cost_i(S')$$

che è proprio quello che stavamo cercando. □

Lemma 2. *Per ogni vettore di strategie S, si ha:*

$$cost(S) \leq \Phi(S) \leq H_k cost(S) \tag{3.10}$$

Dimostrazione.

$$\begin{aligned} cost(S) \leq \Phi(S) &= \sum_{e \in E} c_e H_{k_e(S)} \\ &= \sum_{e \in N(S)} c_e H_{k_e(S)} \leq \sum_{e \in N(S)} c_e H_k \end{aligned} \tag{3.11}$$

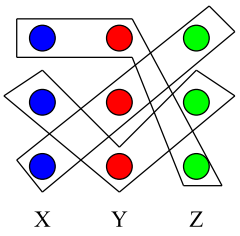
$$1 \leq k_e(S) \leq k \text{ per } e \in N(S) \tag{3.12}$$

□

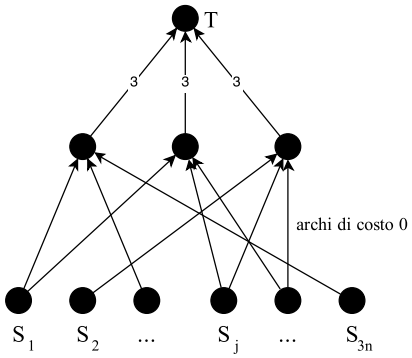
Teorema 5. *Data un'istanza di GCG e un valore C, dimostrare che un gioco ha un equilibrio di Nash di costo al più C è un problema NP-completo.*

Dimostrazione. Riduzione da 3D Matching. Si parte da un'istanza di 3D Matching per generare un'istanza di GCG. □

3.1.8 Problema del Matching 3-dimensionale



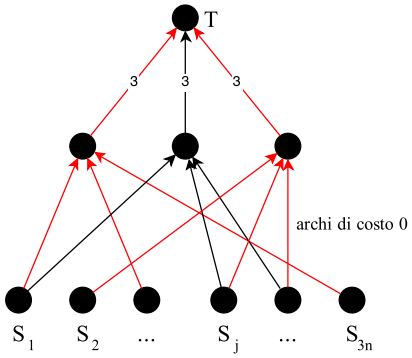
Input: ho tre insiemi disgiunti X, Y, Z di dimensione n, su di essi definisco un insieme di triple ordinate $T \subseteq X \times Y \times Z$.



Richiesta: il problema cerca un sottoinsieme di n triple in T tale che $\forall t \in X \cup Y \cup Z$ è contenuto esattamente in una delle triple.

Posso ridurre il problema 3D-matching ad un GCG costruendo un grafo opportuno:

Esiste una soluzione al problema 3D-matching se e solo se esiste un equilibrio di Nash di costo al più 3n. Supponiamo che esista una soluzione. Definiamo S come il profilo si strategie nel quale ogni giocatore sceglie un cammino che unisce le triple del gruppo prescelto dal 3D-matching, il costo è proprio $cost(S) = 3n$ e S così definito è un equilibrio di Nash.



Ipotizziamo che esista un altro Equilibrio di Nash S' di costo $cost(S') \leq 3n$, $N(S)$ utilizza al più n archi di costo 3 e ogni arco di questo tipo può servire al più 3 giocatori quindi gli archi di costo 3 sono esattamente n e definiscono un insieme di triple che sono soluzione del problema 3D-matching.

3.1.9 Maximum Cut Game

Input: grafo non orientato $G = (V, E)$ i cui nodi sono giocatori egoisti; se $u \in V$ è un nodo del grafo, $S_u \in$ {rosso, verde} è la strategia ad esso associata,

S indica il vettore delle strategie di tutti i nodi.

Richiesta: il singolo giocatore ha come scopo massimizzare il proprio payoff dato da $p_u(S) =\mid \{(u,v) \in E : S_u \neq S_v\} \mid$, il costo del benessere sociale relativo al vettore delle strategie S è dato da

$$\sum_{u \in V} p_u = 2 \cdot \sharp \{\text{archi che appartengono al taglio rosso-verde}\}$$

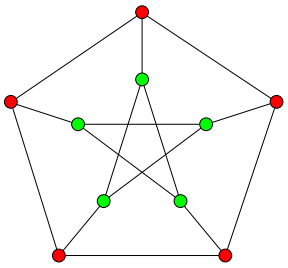


Figura 3.14: Grafo di Petersen

Ora iniziamo ad osservare il nostro problema su un grafo di Petersen e ci chiediamo:

- Esiste sempre un equilibrio di Nash?
- Quanto tale equilibrio può essere lontano dall'ottimo sociale?
- Il gioco riprodotto in modo iterativo converge sempre ad un equilibrio di Nash?

Quello in figura **??** è un esempio di equilibrio di Nash sul grafo di Petersen ottenuto in modo iterativo, il numero di archi che congiungono vertici di colore diverso è $p(S) = 12$.

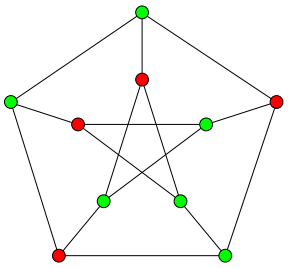


Figura 3.15: Grafo di Petersen in equilibrio

Esercizio

Mostrare che il Max-cut game è un gioco potenziale e mostrare che:

- il PoS vale 1;
- il PoA è $\leq 1/2$;
- esiste un'istanza del gioco in equilibrio di Nash che ha costo sociale pari ad $1/2$ dell'ottimo sociale.

Per mostrare che il Max-cut game ammette una funzione potenziale basta pensare a cosa succede quando un nodo decide di cambiare colore: immaginiamo che i nodi siano divisi in due insiemi a seconda del colore, tra i nodi ci saranno archi interni agli insiemi e archi che collegano nodi di colore diverso tra i due insiemi, il numero di tali archi è quello che vorrei massimizzare con il Max-cut game; consideriamo l'insieme $N(i)$ dei vicini del nodo i rosso in figura **??**, una parte $N(i_r)$ saranno rossi, mentre l'altra parte $N(i_v)$ sono verdi; nel momento in cui i cambia colore, nel conteggio totale degli archi che fanno parte degli archi del taglio rosso-verde si vanno ad aggiungere $\mid N(i_r) \mid$ mentre si devono sottrarre $\mid N(i_v) \mid$ che sono diventati interni all'insieme dei verdi, questa differenza è proprio uguale a $p_i(S') - p_i(S) = \mid N(i_r) \mid - \mid N(i_v) \mid$.

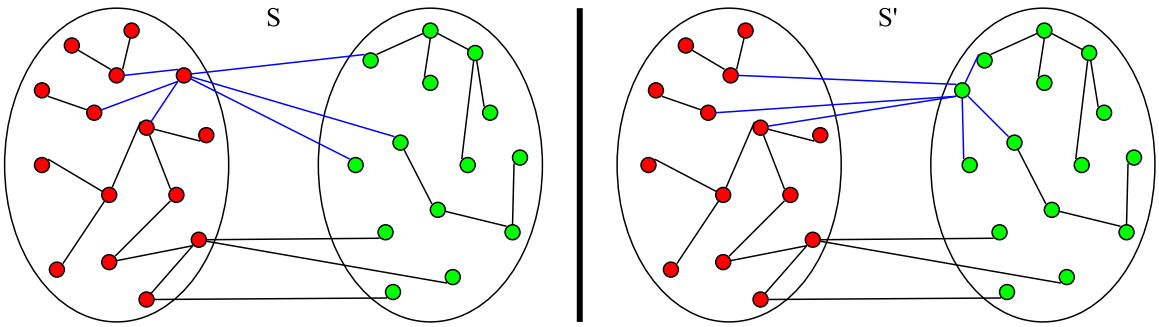


Figura 3.16: Grafo in cui studio cosa succede se un nodo cambia colore.

Per mostrare che il PoS e il PoA valgono rispettivamente 1 e $1/2$, è sufficiente osservare i due grafi in figura **??**

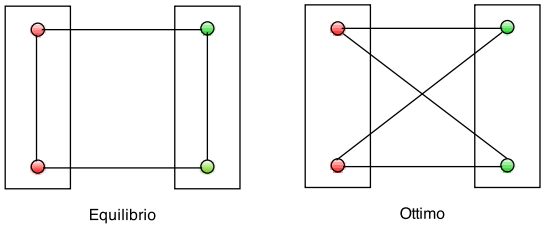


Figura 3.17: Grafo in equilibrio e grafo ottimo

3.2 Local Connection Game

I Local Connection Game modellizzano situazioni in cui un insieme di player concorrono alla costruzione di una rete in modo non cooperativo. A differenza dei Global Connection Game, nei Local (anche chiamati in letteratura Network Formation Game) non esiste alcuna rete iniziale, ma sono i nodi stessi a crearla. Gli n nodi sono i player e la strategia del nodo u consiste nello scegliere un sottoinsieme n_u degli altri nodi per costruire un arco verso di essi e consentire dunque la comunicazione. L'obiettivo è quello di costruire una rete di qualità pagando il meno possibile e minimizzando contestualmente il numero di archi verso gli altri nodi. Schematizzando il modello, si hanno:

- n player, che rappresentano i nodi del grafo da costruire
- Strategia di un player u: selezionare un sottoinsieme qualsiasi di archi non diretti e non pesati a lui incidenti, per costruire la rete.
- Dato un vettore di strategie S, sia $G(S)$ la rete costruita: esiste un arco (u, v) se l'arco è stato comprato dal nodo u, oppure dal nodo v, oppure da entrambi. L'arco una volta acquistato può essere utilizzato da qualsiasi altro nodo.
- Gli obiettivi del player u sono:
  - Minimizzare le distanze verso gli altri nodi
  - Pagare il meno possibile
- Per convenzione i nodi isolati hanno distanza ∞ rispetto a tutti gli altri nodi.

Se ogni arco ha un costo pari ad α, e sia $dist_{G(S)}(u, v)$ la lunghezza del cammino minimo (non essendo un grafo pesato, in termini di numero di archi) tra il nodo u e il nodo v, lo scopo del player u è minimizzare il payoff:

$$cost_u(S) = \alpha n_u + \sum_{v \in G(S)} dist_{G(S)}(u, v) \tag{3.13}$$

dove, come già sottolineato, α è il costo dell'arco, n_u il numero di archi acquistati dal player u (pesati dalla costante α e **incidenti a esso**) e $dist_{G(S)}(u, v)$ il numero di archi del cammino tra i nodi u e v.

Come già sottolineato per i Global Connection Game, anche per quanto riguarda i Local Connection Game è bene specificare che non si tratta di problemi di ottimizzazione poichè la strategia di un player u dipende anche dalle strategie dei restanti player che partecipano al gioco.

Richiami
- Viene utilizzato un equilibrio di Nash come soluzione - Per valutare la qualità di una rete si considererà il costo sociale: somma dei costi di tutti i player - La rete è ottima o socialmente efficiente se minimizza il costo sociale. Lo scopo sarà capire quanto una rete stabile è buona rispetto alla rete ottima. - Un grafo $G(V, E)$ è stabile se esiste un vettore di strategie S tale che S è un equilibrio di Nash.

Esempio. Sia $G(V, E)$ il grafo in figura **??**, dove l'orientamento degli archi serve a denotare quali nodi hanno acquistato gli archi.

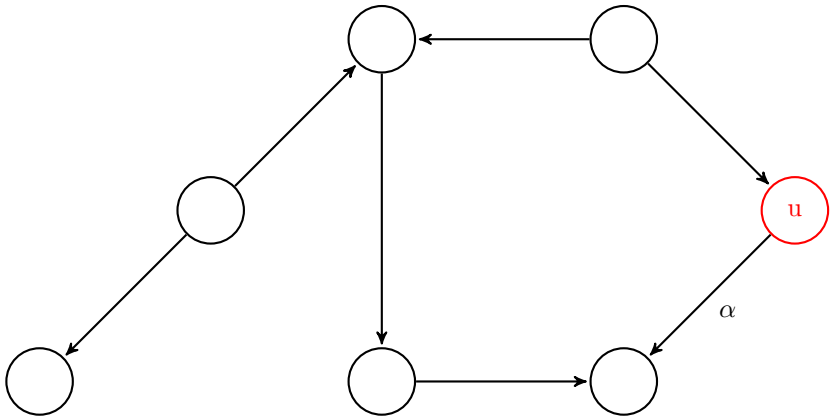


Figura 3.18: G(V, E)

Soffermandoci sul nodo u è possibile notare che tale nodo ha acquistato un solo arco, di un determinato costo α (per non appesantire il grafo, il costo α è stato volontariamente omesso). Poiché nei Location Connection Game oltre alla minimizzazione del costo per l'acquisto degli archi lo scopo di un player è anche quello di minimizzare le distanze tra se stesso e tutti gli altri nodi, nel grafo in figura **??** vengono riportate le distanze per il nodo u.

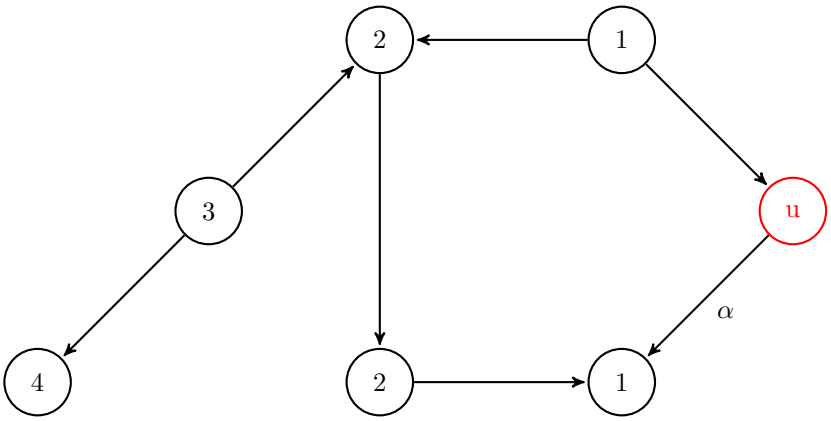


Figura 3.19: G(V, E)

Dove si evince chiaramente che:

$$c_u = \alpha + 13$$

E' una rete in equilibrio? Il nodo u potrebbe, ad esempio, acquistare un arco verso il nodo a distanza 4 per diminuire la distanza da esso e dal nodo di costo 3, come illustrato in figura **??**.

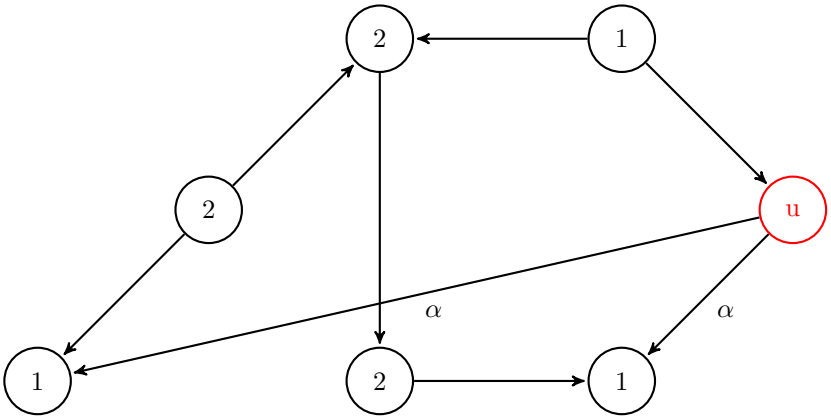


Figura 3.20: G(V, E)

In questo caso invece:

$$c_u = 2\alpha + 9$$

E' chiaro che non si può affermare che questa seconda soluzione sia migliore rispetto alla prima, dal momento che il payoff dipende fortemente dal parametro α.

Esempio. Sia $G(V, E)$ il grafo in figura **??**, dove $\alpha = 5$.

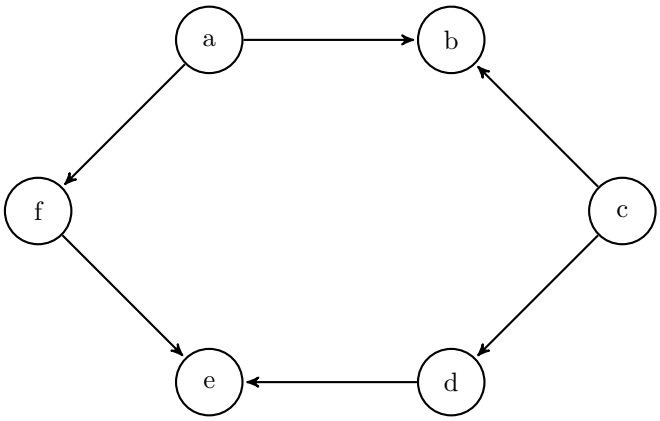


Figura 3.21: G(V, E)

Si prenda in considerazione il nodo d, nella configurazione riportata il payoff per il nodo, come anche illustrato in **??**, è pari a:

$$c_d = 5 + 9 = 14$$

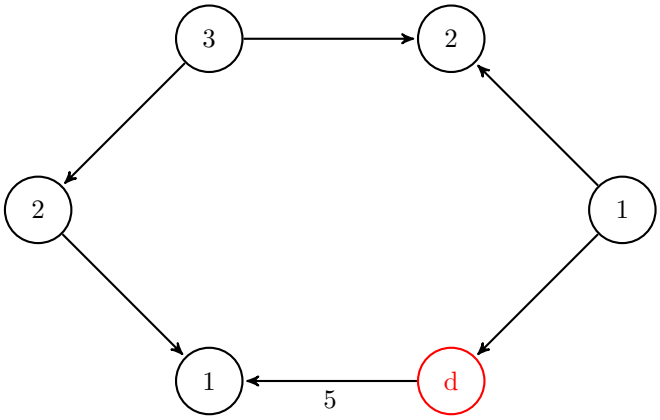


Figura 3.22: G(V, E)

(**Passo 1**). Poiché non è una rete stabile, il nodo d può cambiare strategia, ad esempio acquistando l'arco (d, a) (figura **??**). In questa nuova situazione:

$$c_d = 2 * 5 + 7 = 17$$

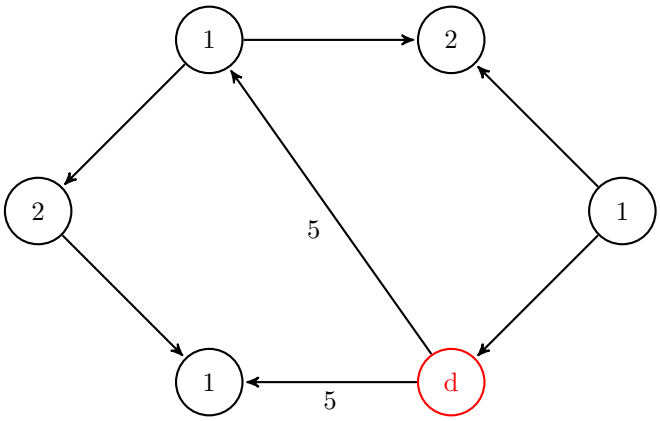


Figura 3.23: G(V, E)

(**Passo 2**). Ancora non è una rete stabile, il nodo d infatti ha aumentato il suo payoff rispetto alla configurazione precedente. Potrebbe quindi cambiare strategia eliminando l'arco (d, e) (figura **??**). In questa nuova situazione:

$$c_d = 0 + 15 = 15$$

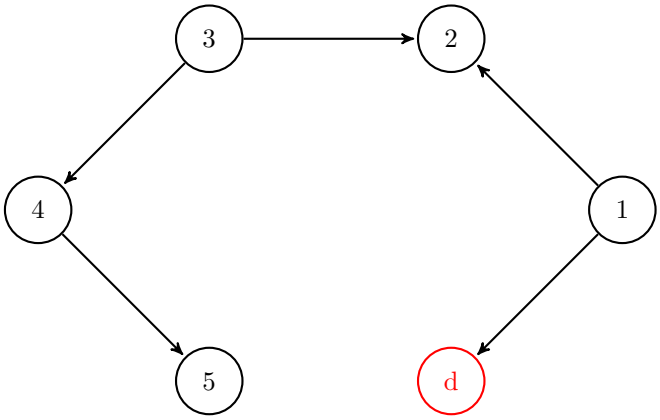


Figura 3.24: G(V, E)

(**Passo 3**). Il nodo d ha diminuito il suo payoff rispetto alla soluzione precedente, ma il costo è ancora superiore rispetto alla soluzione iniziale. Potrebbe quindi adottare una nuova strategia: eliminare l'arco (d, e) e acquistare l'arco (d, f) (figura **??**). In questa nuova situazione il nodo d non ha alcuna convenienza a cambiare strategia, dal momento che:

$$c_d = 5 + 6 = 13$$

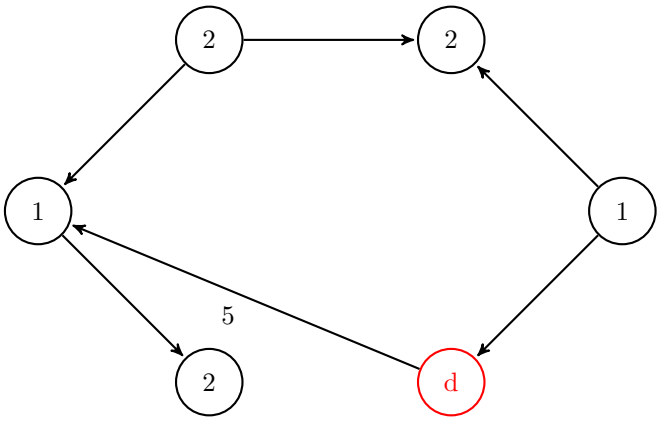


Figura 3.25: G(V, E)

(**Passo 4**). La rete però non è ancora stabile. Il nodo c può infatti migliorare il suo payoff eliminando gli archi (c, b) e (c, d) e acquistando contestualmente l'arco (c, f) (figura **??**). In questa nuova situazione:

$$c_c = 5 + 10 = 15$$

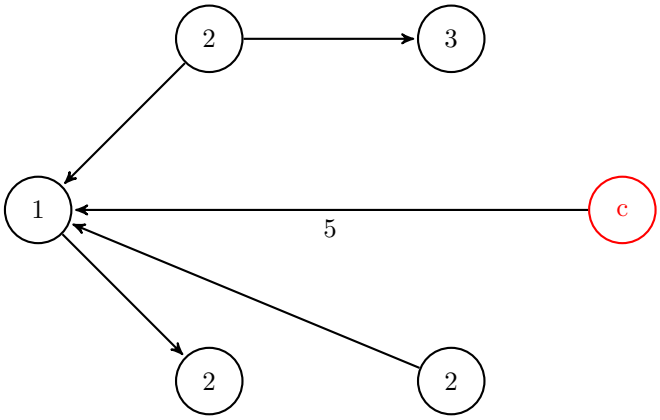


Figura 3.26: G(S)

A questo punto la rete è stabile, i restanti nodi infatti non hanno alcuna convenienza nel cambiare strategia.

Osservazioni. Gli esempi sopra riportati dovrebbero essere utili anche per trarre alcune semplici conclusioni. Prima di tutto si noti che una rete non può mai essere disconnessa poichè ciò significherebbe che almeno un player sta pagando ∞, che è ben superiore rispetto ad un qualsiasi α. Inoltre in una rete stabile un arco non è mai acquistato da due nodi, dal momento che lo stesso arco, acquistato da un determinato nodo, può essere utilizzato da tutti i player che partecipano al gioco. Infine si sottolinea che nel costo sociale la distanza $d_{G(S)}(u, v)$ tra una coppia di nodi (u, v) che utilizza lo stesso arco è conteggiata due volte, poichè $d_{G(S)}(u, v) = 1 = d_{G(S)}(v, u)$, quindi $d_{G(S)}(u, v) = 2$. Ne consegue che il costo sociale di una rete stabile $G(S) = (V, E)$ è:

$$SC(S) = SC(G) = \alpha|E| + \sum_{u,v} dist_{G(S)}(u, v) \tag{3.14}$$

3.2.1 Studio dell'ottimo

Teorema 6. *Se $\alpha \leq 2$ allora il grafo completo (figura **??**) è una soluzione ottima, se invece $\alpha \geq 2$ allora ogni stella (figura **??**) è una soluzione ottima.*

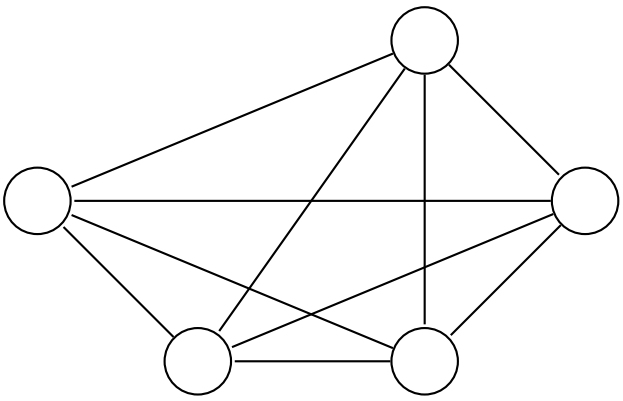


Figura 3.27: G(V, E): Grafo completo

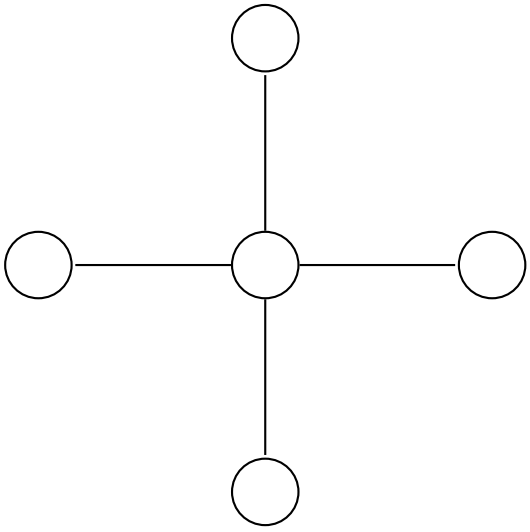


Figura 3.28: G(V, E): Stella

Dimostrazione. Sia $G(V, E)$ una soluzione ottima e $SC(G) = OPT$, per dimostrare tale teorema si consideri una soluzione con $|E| = m$ archi. Si parta dal presupposto che $m \geq n - 1$, altrimenti il grafo sarebbe disconnesso e ci sarebbe almeno un nodo a distanza ∞ e, come già osservato precedentemente, ciò non è mai possibie.
Le coppie di nodi a distanza 1 sono:

$$\text{coppie ordinate di nodi a distanza } 1 \ = 2m \tag{3.15}$$

Le coppie di nodi non direttamente connesse (si ricorda che sono coppie ordinate, e quindi ogni nodo contribuisce 2 volte), ovvero a distanza almeno 2, sono (tolte quelle a distanza 1):

$$\text{coppie ordinate di nodi a distanza almeno 2 } = n(n-1) - 2m \tag{3.16}$$

Sostituendo le due equazioni nell'equazione del costo sociale si ottiene il Lower Bound (LB) in funzione degli archi dell'ottimo:

$$\begin{aligned} OPT \geq \alpha m + 2m + 2(n(n-1) - 2m) = \\ (\alpha - 2)m + 2n(n-1) = \\ LB(m) \end{aligned} \tag{3.17}$$

Dove α è il costo di acquisto degli archi, mentre il numero di archi a distanza almeno 2 va moltiplicato per 2 dal momento che il costo è rappresentato dalla distanza stessa tra le coppie di nodi.

Importante: Si noti che il Lower Bound trovato è un limite inferiore sia per la stella dove il numero di archi è pari a $|E| = m = n - 1$ che per il grafo completo dove invece $|E| = m = n(n-1)/2$.

$$\begin{aligned} OPT \geq min_m LB(m) \geq \\ (\alpha \leq 2 \rightarrow \max m) : LB(n(n-1)/2) = SC(K_n) \\ (\alpha \geq 2 \rightarrow \min m) : LB(n-1) = SC \text{ di ogni stella} \end{aligned} \tag{3.18}$$

Con $\alpha = 2$ il costo sociale della stella e del grafo completo è uguale.

□

3.2.2 Studio dell'equilibrio e PoS

La soglia che di quilibrio per i due grafi è diversa dalla soglia dell'ottimo.

Teorema 7. *Se $\alpha \leq 1$ il grafo completo è stabile, mentre se $\alpha \geq 1$ allora il grafo a stella è stabile.*

Dimostrazione.

- **Caso $\alpha \leq 1$:** Sia $G(V, E)$ un grafo completo. Si prenda in considerazione un player u qualsiasi, questo non compra nuovi archi per definizione di grafo completo e ne consegue che l'unica strategia che può attuare consiste nel rimuovere k archi. Il player u risparmierebbe, rimuovendo k archi, k volte α (con $\alpha \leq 1$) ma la distanza tra se stesso e gli altri k nodi salirebbe da 1 a 2, che è maggiore del risparmio dovuto alla rimozione degli archi.

- **Caso $\alpha \geq 1$:** Sia $G(V, E)$ un grafo a stella. Supponiamo che il player c al centro della stella acquisti tutti gli archi. E' evidente che il player c non può rimuovere alcun arco poiché altrimenti disconnetterebbe il grafo

e la distanza tra se stesso e il nodo da cui si è disconnesso sarebbe pari a ∞. D'altro canto ad un qualsiasi player u non conviene acquistare nuovi k archi dal momento che pagherebbe in più $\alpha * k$ risparmiando solo k in distanza (pari alla distanza tra se stesso e i nodi per cui ha comprato i nuovi k archi).

□

Teorema 8. *Se $\alpha \leq 1$ o $\alpha \geq 2$ allora il prezzo della stabilità PoS è pari a 1. Per $1 < \alpha < 2$, il prezzo della stabilità è al più 4/3.*

Dimostrazione.

- **Caso $\alpha \leq 1$, $\alpha \geq 2$:** l'equilibrio coincide con l'ottimo e, visto che non esistono altri equilibri, il miglior equilibrio di Nash è anche unico. La soluzione si intuisce immediatamente dai lemmi precedenti.
- **Caso $1 < \alpha < 2$:** Sia T un grafo a stella che, come già visto, è un equilibrio di Nash per $\alpha \geq 1$, e sia K_n il grafo completo che, invece, è la soluzione ottima per $\alpha \geq 2$. Il prezzo della stabilità è pari a:

$$\begin{aligned} PoS &\leq \\ \frac{SC(T)}{SC(K_n)} &= \\ \frac{(\alpha-2)(n-1)+2n(n-1)}{\alpha n(n-1)/2+n(n-1)} &\leq \\ \frac{-1(n-1)+2n(n-1)}{n(n-1)/2+n(n-1)} &= \\ \frac{2n-1}{(3/2)n} &= \\ \frac{4n-2}{3n} &< \\ \frac{4}{3} & \end{aligned} \tag{3.19}$$

□

Schema riassuntivo

Dai lemmi precedenti si può ricavare lo schema riassuntivo in figura **??**.

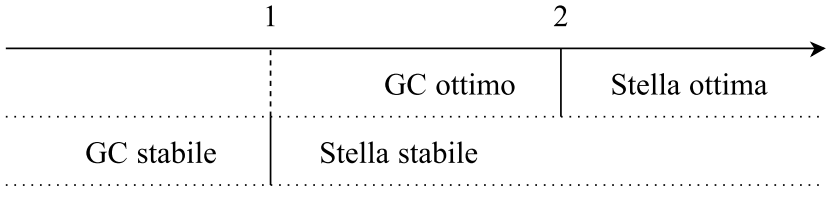


Figura 3.29: Riassunto del LCG in base al costo degli archi costruiti.

3.2.3 Studio dell'equilibrio e PoA

Sia $G(V, E)$ un grafo, si definisce **diametro** di G la massima distanza tra due nodi. Nel grafo rappresentato in figura **??** il diametro è pari a 2, mentre nel grafo in figura **??** il diametro è pari a 4.

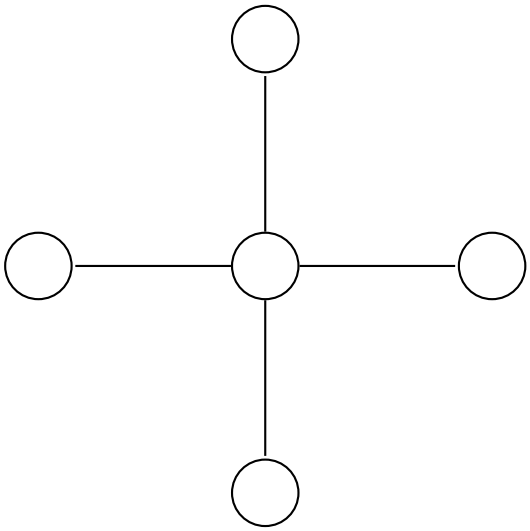


Figura 3.30: G(V, E)

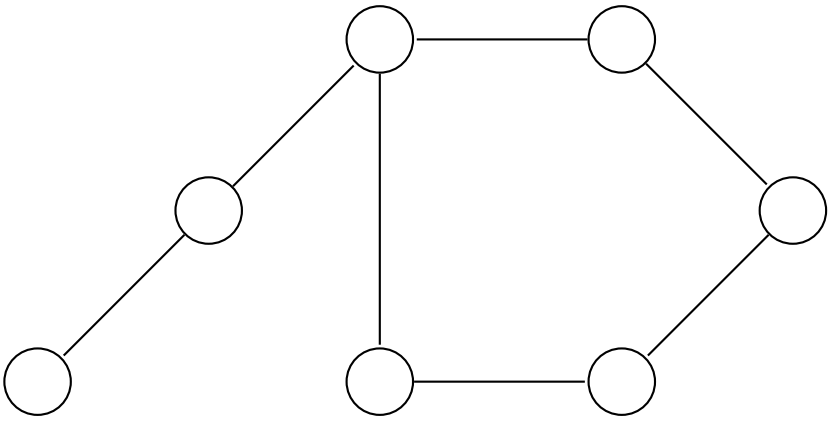


Figura 3.31: G(V, E)

Sia $G(V, E)$ un grafo, un arco e è un **cut edge** o arco del taglio di G se $G - e = (V, E \backslash e)$ è un grafo disconnesso.
Un grafo con n nodi e m archi ha al più $n - 1$ cut edge.

Lemma 3. *Il diametro di una rete stabile è al più* $2\sqrt{\alpha} + 1$

Dimostrazione. Preso $G = (V, E)$, un grafo stabile, consideriamo il cammino più breve che unisce due nodi $u, v \in V$.

Naturalmente $\exists k$ tale che $2k \leq dist_G(u, v) \leq 2k + 1$, quindi u e v sono separati da un certo numero di archi ($2k$ se pari, $2k + 1$ se dispari). I nodi nel cammino saranno a distanze crescenti da 1 a $2k$ (o $2k + 1$) da u. Immaginiamo

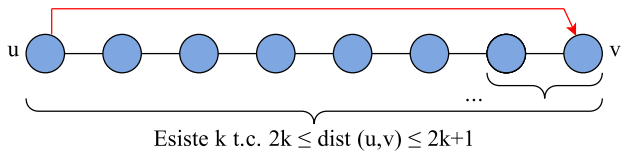


Figura 3.32: Cammino minimo tra due nodi del grafo.

di costruire l'arco (u, v), allora le distanze da u verso gli altri nodi cambieranno in questo modo:

- Il nodo a distanza $2k$ ora è a distanza 1 miglioramento di $2k - 1$;
- Il nodo a distanza $2k - 1$ ora è a distanza 2 miglioramento di $2k - 3$;
- ...
- Il nodo a distanza $k + 1$ ora è a distanza k miglioramento di 1.

Quindi il miglioramento della rete è $\geq \sum_{i=0}^{k-1}(2i + 1) = k^2$. Se la rete è stabile significa che non vale la pena costruire un arco (u, v), cioè α è maggiore del miglioramento delle distanze tra u e in nodi del cammino. Quindi

$$\alpha \geq k^2, k \leq \sqrt{\alpha}$$

ricordando che

$$2k \leq dist_G(u, v) \leq 2k + 1$$

allora

$$dist_G(u, v) \leq 2\sqrt{\alpha} + 1$$

□

Lemma 4. *Sia G una rete con diametro d, e sia $e = (u, v)$ un non cut-edge. Allora in $G - e$, ogni nodo w aumenta la sua distanza da u al più di $2d$.*

Lemma 5. *Sia G una rete stabile, e sia F l'insieme dei non cut-edge acquistato dal nodo u. Allora $|F| \leq (n - 1)2d/\alpha$*

Dimostrazione. Costruiamo l'albero dei cammini minimi BFS$(u) = T$ (figura **??**

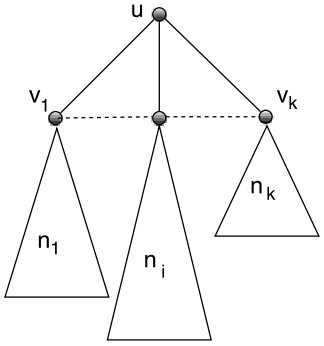


Figura 3.33: T

Poiché ogni arco (u, v_i) è un non cut-edge, se u rimuove uno di questi archi, i nodi contenuti nei rispettivi sottoalberi non si allontanano di al più $2d$ (come osservato nel lemma precedente) da u. Dal momento che, però, G è una rete stabile, u non deve avere alcuna convenienza nell'eliminare l'arco e, quindi, deve valere:

$$\alpha \leq n_i 2d$$

E quindi:

$$k\alpha = \sum_{i=1}^{k} \alpha \leq \sum_{i=1}^{k} n_i 2d \leq (n-1)2d \tag{3.20}$$

In definitiva:

$$k \leq \frac{(n-1)2d}{\alpha} \tag{3.21}$$

□

Lemma 6. *Il costo sociale SC di una qualsiasi rete stabile con diametro d è al più $O(d)$ volte l'ottimo sociale.*

Dimostrazione. Ricordiamo che:

$$OPT \geq \alpha(n-1) + n(n-1)$$

Si noti che (dalla formula del costo sociale)

$$\alpha|E| = \alpha|E_{cut-edge}| + \alpha|E_{non-cut-edge}| \leq \alpha(n-1) + n(n-1)2d \leq 2d * OPT \tag{3.22}$$

Dove $n(n-1)2d$ deriva direttamente dal lemma precedente.
E inoltre:

$$SC(G) = \alpha|E| + \sum_{u,v} d_G(u,v) \leq 2d * OPT + d * OPT = 3d * OPT \tag{3.23}$$

□

Lemma 7. *Il prezzo dell'anarchia PoA è al più* $O(\sqrt{\alpha})$.

Dimostrazione. Si evince chiaramente dalle precedenti dimostrazioni:

$$\frac{3d * OPT}{OPT} = \frac{3\sqrt{\alpha} * OPT}{OPT} = O(\sqrt{\alpha}) \tag{3.24}$$

□

3.2.4 Riduzione da Dominating Set

Teorema 9. *Date le strategie degli altri player, calcolare la best response di un player dato è un problema NP-Hard.*

Dimostrazione. Si effettua la riduzione dal Dominating Set. Sia $G(V, E)$ un grafo, si vuole trovare un sottoinsieme dei nodi $U \subset C$ tale che per ogni nodo $v \in V - U$ esiste un nodo $u \in U$ per cui esiste un arco $(u, v) \in E$.

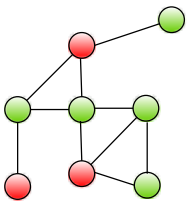


Figura 3.34: G(V, E)

Claim: La strategia di un player i ha un costo $\leq \alpha k + 2n - k$ se e solo se c'è un Dominating Set di dimensione $\leq k$.

Dimostrazione $\leftarrow$
Dato un Dominating Set U di dimensione k, il player i compra tutti archi incidenti ai nodi in U. Il costo per il player è:

$$cost_i = \alpha k + 2(n - k) + k = \alpha k + 2n - k$$

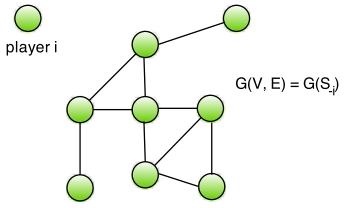


Figura 3.35: G(V, E)

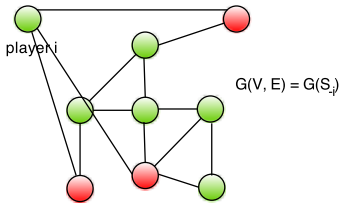


Figura 3.36: G(V, E)

Dimostrazione →
Sia S_i la strategia che fornisce un costo $\leq \alpha k + 2n - k$.

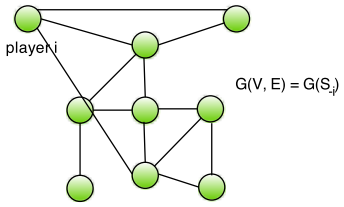


Figura 3.37: G(V, E)

Si modifica S_i in questo modo:

1. Se c'è un nodo v tale che la distanza da x è ≥ 3 in $G(S)$, allora aggiungi l'arco (x, v) a S_i, riducendo il costo.
2. Ripeti finchè la distanza tra x e ogni altro nodo è pari a 1 o 2.

Sia U l'insieme dei nodi a distanza 1 da x, questo rappresenta a tutti gli effetti un Dominating Set del grafo originale. Il costo per il player i è infatti:

$$cost_i(S) = \alpha|U| + 2n - |U| \leq \alpha k + 2n - k$$
$$|U| \leq k$$

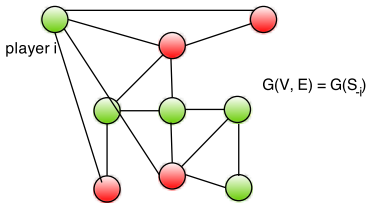


Figura 3.38: G(V, E)

□

Capitolo 4

Mechanism Design

I Mechanism Design sono una sottoarea della teoria dei giochi il cui tipo di studio - rispetto alla prima parte - ha un aspetto più algoritmico e progettuale, presente in scenari non cooperativi. Informalmente di base il problema si può vedere in tal modo: si osserva la realtà e, in virtù di questa osservazione, si arriva alla conclusione che si tratta di una realtà 'complicata' anche per via del fatto che gli agenti protagonisti sono agenti egoisti. Ad esempio, nella città di Roma, è facile osservare spesso il Grande Raccordo Anulare bloccato dal traffico, oppure il continuo inquinamento delle industrie nei territori della Provincia. Sono scenari ordinari, anche se ciò che una persona vorrebbe sono, sempre facendo riferimento agli stessi esempi, le strade libere e la vita di campagna, in cui si sfruttano le energie alternative invece del petrolio e del carbone.
Sorge quindi una domanda: come è possibile, in presenza di agenti egoisti non coordinati, arrivare a situazioni del secondo tipo? Si potrebbe pensare di fornire degli incentivi, ovvero incentivare gli agenti a comportarsi in modo più sociale possibile, aumentando i mezzi pubblici, o fornendo degli aiuti economici per l'acquisto di pannelli solari, e via discorrendo.

Tutto ciò è alla base dei Mechanism Design: affinchè un sistema raggiunga un determinato obiettivo, in presenza di agenti che hanno obiettivi differenti, deve incentivare attraverso pagamenti gli agenti per convincerli a cooperare e raggiungere l'obiettivo che si è prefissato esso stesso. L'obiettivo che il sistema si prefigge deve essere confrontato con le preferenze individuali dei singoli individui e spesso tali preferenze sono private, come nel caso del problema della scelta di una legge elettorale, e possono essere utilizzate dagli stessi per manipolare il sistema. Di fatto questi problemi hanno come input solo parte delle informazioni, ovvero il sistema vorrebbe ottimizzare una data funzione obiettivo che dipende dagli interessi dei singoli individui che detengono informazioni che il sistema non conosce.
Esiste dunque un meccanismo che può portare, tramite incentivi economici, gli agenti egoisti a comportarsi in modo tale che l'obiettivo desiderato sia pienamente raggiunto? La risposta, come si vedrà a breve, è positiva.
Il meccanismo, infatti, riuscirà a definire un gioco in cui i player hanno interesse a collaborare con il sistema. Chiariamo le idee con un esempio:
Esempio dell'asta Il sistema vuole dare un determinato oggetto di valore ad uno dei partecipanti dell'asta. L'asta ha tale funzionamento:

- L'asta è un Gioco. Comprende quindi n player che devono scegliere una strategia allo scopo di ottimizzare il proprio payoff.
- La strategia di ogni giocatore consiste nel dichiarare un valore r_i. Questa scelta viene fatta all'oscuro delle scelte degli altri player.
- Ogni giocatore detiene privatamente l'informazione sulla propria valutazione dell'oggetto. Questa informazione è chiamata **tipo** e si denota con t_i.
- Il payoff di ogni giocatore è 0 se non ottiene l'oggetto, oppure $u_i = t_i - p$ se vincitore dell'asta.
- Il sistema non conosce i tipi privati dei giocatori.
- Il sistema vuole che ogni giocatore dichiari come valore il proprio tipo privato.
- Il sistema deve stabilire le regole del gioco: chi sarà il vincitore e il valore p che il vincitore dovrà pagare.
- I player conoscono le regole del gioco.

Simulazione del gioco. Preso atto delle regole del gioco (il meccanismo è fissato), i player faranno un'offerta che non necessariamente corrisponderà al loro *tipo*. Se il player i non riceve l'oggetto di valore, il suo payoff sarà pari a zero. Se invece il player i vince l'asta, e gli viene chiesto di pagare p, allora il suo utile - che può essere visto anche come forma di contetezza - è pari a quanto lui valutava realmente l'oggetto meno ciò che dovrà pagare. Ovvero, in questa tipologia di giochi lo scopo dei player è, come si evince chiaramente, massimizzare la differenza tra t_i e p.

Lo scopo quindi, in termini informatici, consiste nell'implementare un algoritmo che trova il massimo tra tutti i t_i senza però conoscere tali valori. E' però possibile creare un meccanismo che incentivi i player a rivelare le informazioni private per estrapolare il massimo, impedendo ai player di dichiarare il falso (sovradichiarando o sottodichiarando).

Esempio dell'asta - Primo meccanismo
Regole:

- Vince chi effettua l'offerta più alta. Il vincitore ottiene l'oggetto di valore.
- Non c'è alcun pagamento.

I player effettuaranno un'offerta cercando di massimizzare il proprio utile (che, viene ricordato, consiste nella differenza tra la valutazione dell'oggetto da parte del player e il pagamento che dovrà fare). Non pagando nulla, in questo caso, l'utile consisterà solo nella valutazione dell'oggetto da parte del player.

Dichiarazioni dei player Si parte dal presupposto che tutti i player vogliono l'oggetto di valore poichè se non lo ottengono avranno un payoff pari a zero.

Conoscendo le regole del gioco, tutti i player offriranno inevitabilmente il massimo, che è pari a infinito. A chi quindi il sistema assegnerà l'oggetto di valore? E' evidente che tale meccanismo non funziona.

Esempio dell'asta - Secondo meccanismo
Regole:

- Vince chi effettua l'offerta più alta. Il vincitore ottiene l'oggetto di valore.
- Il vincitore deve pagare quanto dichiarato nell'offerta.

Dichiarazioni dei player. I player non dichiarano più il massimo, nè dichiarano un prezzo maggiore rispetto ai rispettivi t_i dal momento che la loro utiltà sarebbe negativa. Rispetto al tipo t_i, le dichiarazioni r_i saranno dunque strettamente minori. La differenza tra t_i e r_i dipende dalla propensione al rischio che ha il singolo player. Dato tale meccanismo, il sistema riuscirà ad effettuare la scelta giusta? No, supponiamo infatti che lo scenario sia il seguente:

$$t_1 = 10, r_1 = 9$$
$$t_2 = 12, r_2 = 8$$
$$t_3 = 7, r_3 = 6$$

Il sistema decreterà vincitore il player 1, che però non rappresenta quel player in grado di valorizzare al massimo l'oggetto di valore vinto. E' evidente che tale meccanismo non funziona, poiché è facilmente manipolabile.

4.1 Vickery Auction

4.1.1 Vickery Auction: versione di massimizzazione

Esiste un meccanismo in grado di impedire ai player di manipolare il sistema: è l'asta di Vickery. *Regole* del meccanismo:

- Vince chi effettua l'offerta più alta. Il vincitore ottiene l'oggetto di valore.
- Il vincitore deve pagare la seconda migliore offerta.

Con tali regole, al fine di massimizzare l'utilità al player conviene dichiarare il vero, indipendentemente dalle scelte degli altri player, ovvero il tipo t_i. Supponiamo che lo scenario sia il seguente:

$$t_1 = 10, r_1 = 10$$
$$t_2 = 12, r_2 = 12$$
$$t_3 = 7, r_3 = 7$$

Il player 2 vince l'oggetto di valore, e paga la seconda migliore offerta, ovvero 10. L'utilità è pari a:

$$u_2 = r_2 - p = t_2 - r_1 = t_2 - t_1 = 2$$

Teorema 10. *Nell'asta di Vickery, per ogni player i,*

$$r_i = t_i$$

è una strategia dominante.

Dimostrazione. Fissati i player i e i relativi t_i, si considerino le strategie del player i. Sia $R = max_{j \neq i} r_j$.

Caso 1: $t_i \geq R$ (R ovviamente è sconosciuto al player i).
- Dichiarando la verità, $r_i = t_i$, il player i vince l'oggetto, e l'utilità è pari a $u_i = t_i - R \geq 0$ (se $t_i = R$ allora il player può vincere o perdere, in base ai meccanismi implementati in caso di parità, ma in entrambi i casi la sua utilità è sempre pari a 0).
- Dichiarando $r_i > R$ con $r_i \neq t_i$, il player i vince l'oggetto e l'utilità è pari a $u_i = t_i - R \geq 0$, ovvero è uguale al caso precedente.
- Dichiarando $r_i < R$, il player i perde l'asta e la sua utilità è pari a $u_i = 0$.

Caso 2: $t_i < R$
- Dichiarando la verità, $r_i = t_i$, il player i perde l'asta e la sua utilità è pari a $u_i = 0$.
- Dichiarando $r_i < R$, con $r_i \neq t_i$, il player i perde l'asta e l'utilità è pari a $u_i = 0$.
- Dichiarando $r_i > R$, il player i vince l'oggetto ma l'utilità è pari a $u_i = t_i - R < 0$

In ogni caso, effettuando dichiarazioni false non si dà luogo a utilità migliori, quindi dichiarare la verità è la strategia dominante.

□

4.1.2 Vickery Auction: versione di minimizzazione

Supponiamo vi sia un sistema il cui scopo è affidare un job ad una delle n macchine disponibili, le n macchine hanno un costo, in termini di tempo impiegato per processare il job. Se una determinata macchina viene selezionata, riceve un pagamento p e la sua utilità è pari a:

$$u_i = p - t_i \tag{4.1}$$

Le *regole* sono le seguenti:

- Vince chi effettua l'offerta più bassa, ossia chi impiega meno per processare il job. Il vincitore ottiene il job.
- Il vincitore riceve la seconda migliore offerta (più bassa).
- t_i non sono ricavi, ma costi
- I pagamenti non vengono effettuati dagli agenti al sistema, ma dal sistema agli agenti

Anche in questo caso ai player conviene dichiarare il vero, indipendentemente dalle scelte degli altri player, ovvero dichiarare nell'offerta quanto costa effettivamente processare il job, quindi il tipo t_i. Supponiamo che lo scenario sia il seguente:

$$t_1 = 10, r_1 = 10$$
$$t_2 = 12, r_2 = 12$$
$$t_3 = 7, r_3 = 7$$

Vince il player 3, che effettua il lavoro a costo $r_3 = t_3 = 7$ e viene pagato $r_1 = t_1 = 10$. La sua utilità è quindi pari a: $u_3 = p - r_3 = p - t_3 = r_1 - t_3 = t_1 - t_3 = 10 - 7 = 3$

4.2 Mechanism Design: gli ingredienti

- I player sono N, ogni player i ha delle informazioni private $t_i \in T_i$ chiamate tipi.
- Vi è un insieme di outcome ammissibili.
- Il sistema, tra gli outcome ammissibili, vuole selezionare uno specifico outcome $f(t)$, dove f è la funzione sociale, che dipende dai tipi privati degli agenti.
- Ogni agente ha uno spazio di strategie S_i, che consiste nella rivelazione di un certo valore r_i (non per forza vale $r_i = t_i$).
- Lo scopo degli agenti è massimizzare l'utile, definito come la differenza tra il pagamento ricevuto e il costo che sostengono in un certo outcome, tale costo è chiamato valutazione. La valutazione di un player i dipende dal suo tipo t_i e dall'outcome che potenzialmente può essere scelto. Ovvero $v_i(t_i, x)$ dice, in termini di costi, che costo ha il player i se il suo tipo è t_i nel momento in cui viene selezionato l'outcome x.

4.2.1 Mechanism Design: gli ingredienti di Vickery Auction

Nella Vickery Auction $v_i(t_i, x)$ è pari esattamente a t_i se l'outcome afferma che ha vinto il player i, altrimenti 0. L'utilità del player i, ciò che in sostanza vuole massimizzare (nel caso della versione di minimizzazione), è:

$$u_i(t_i, x) = p_i(x) - v_i(t_i, x) \tag{4.2}$$

Il player vuole massimizzare l'utile, nell'outcome x l'utile è definito come il pagamento al player i meno un costo che il player sostiene in x. In un certo senso l'utile del player dipende dal pagamento (che non è correlato con t_i perchè il meccanismo non lo conosce) ma soprattutto dipende dal costo che il player sostiene, e tale costo è legato al suo tipo.

Esempio: Se il costo di un agente per un job è 80 e viene pagato 100 per effettuare tale job, la sua utilità è pari a 20.

4.2.2 Mechanism Design: obiettivi

L'obiettivo è implementare una funzione di scelta sociale come ad esempio la progettazione di un meccanismo $M =< g(r), p(x) >$ dove:

- $g(r)$ è un algoritmo che calcola un outcome (in funzione delle dichiarazioni dei player)
- $x = g(r)$ è una funzione dei tipi riportati dai player, gli r_i
- $p(x)$ è uno schema di pagamento valutato in base ad ogni outcome x.

Il pagamento dipende dall'outcome x, che dipende da ciò che dichiarano i player i, e quindi anche il pagamento dipende da ciò che dichiarano i player i. In conclusione si vuole fare in modo che, quando gli agenti dichiarano un determinato $r_i \neq t_i$, l'esecuzione dell'algoritmo restituisce come risultato lo stesso che avrebbe restituito nel caso in cui l'algoritmo fosse stato eseguito su t_i.

Ricapitolando:

- Gli N agenti sono i player
- Il meccanismo stabilisce le regole del gioco, la matrice dei payoff è data in forma implicita ed è definita dalle utilità dei player in funzione dei tipi e degli outcome.

4.2.3 Mechanism Design con strategie dominanti

Un meccanismo $M =< g(), p() >$ realizza la funzione sociale con strategie dominanti se esiste $r^* = (r_1^*, r_2^*, ..., r_N^*)$, ovvero un equilibrio di strategie dominanti, tale che $g(r^*) = f(t)$. Dunque se un player i gioca r_i^*, questa è la migliore strategia indipendentemente da ciò che giocano gli altri player. In questa tipologia di giochi, se il player i dichiara r_i^* e gli altri player dichiarano un r_{-i} qualsiasi, l'utilità del player i con un determinato t_i è maggiore o uguale dell'utilità sull'outcome che l'algoritmo calcola quando il player dichiara, al posto di r_i^*, un r_i qualsiasi. r_i^* è sempre la soluzione migliore, perchè la sua utilità è maggiore rispetto a tutti gli outcome modificando la sua dichiarazione. Ovvero:

$$u_i(t_i, g(r_{-i}, r_i^*)) \geq u_i(t_i, g(r)) \tag{4.3}$$

Dove $g(r_{-i}, r_i^*) = g(r_1, ..., r_{i-1}, r_i^*, r_{i+1}, ..., r_N)$.
Se truth telling è la strategia dominante, allora il meccanismo è chiamato truthful ovvero:

$$r^* = t \tag{4.4}$$

E quindi i player tendono a dichiarare i propri tipi invece di manipolare il gioco e, inoltre, l'algoritmo del meccanismo viene eseguito su input veri del problema.

4.3 Esempi

L'asta del vino Supponiamo vi sia un'asta in cui il sistema vuole piazzare k bottiglie di vino pregiato. *Regole*:

- Ciascun player può vincere una sola bottiglia di vino.
- t_i rappresenta quanto un player è disposto a pagare per una bottiglia di vino
- Vi sono k vincitori, ovvero il sistema deve selezionare un sottoinsieme di player di cardinalità esattamente k, $|k|$
- Tra tutti gli outcome possibili, il sistema deve scegliere l'outcome con valore più alto, quindi si tratta di un problema di massimizzazione.
- Utilità del player i: $u_i = t_i - p$.

Costruzione di un ponte Supponiamo vi sia una nazione che deve decidere se costruire o meno un nuovo ponte. *Regole*:

- L'opera ha un costo C
- I cittadini o player i forniscono un valore intrinseco al ponte, t_i, che rappresenta di fatto quanto un cittadino i-esimo valuta la convenienza di avere o meno il ponte (quanto sarebbe disposto a pagare).
- Funzione obiettivo del sistema: il ponte viene costruito se i cittadini valutano l'opera abbastanza rispetto al costo reale di costruzione, quando $\sum_i t_i > C$
- Utilità del player i: $u_i = t_i - p_i$.

Commercio bilaterale Supponiamo vi siano un venditore e un compratore, il venditore s è un soggetto che ha intenzione di vendere la sua automobile mentre il compratore b è un soggetto che ha intenzione di acquistare una nuova automobile. Il meccanismo, in questo caso, è un mediatore. *Regole*:

- Tipo del venditore t_s rappresenta il valore dell'automobile secondo il venditore
- Tipo del compratore t_b rappresenta il valore dell'automobile secondo il compratore
- Il meccanismo deve decidere se la vendita deve avvenire o meno, $F =$ scambio, non scambio.
- La vendita avverrà solo se l'automobile passa da chi la valuta di meno a chi la valuta di più, quindi $t_b > t_s$
- Se lo scambio avviene, il meccanismo paga il venditore p_s mentre ottiene dal compratore p_b.
- Per fare in modo che via sia la truthfulness, in modo tale che compratore e venditore dichirino i loro tipo $t_s = r_s$ e $t_b = r_b$, il meccanismo deve pagare una determinata quantità.
- Utilità del venditore s: $u_s = p_s - t_s$. Utilità del compratore b: $u_b = t_b - p_b$.

Acquisto di un arco in una rete Supponiamo vi sia un grafo $G(V, E)$ pesato. Ogni arco $e \in E$ è controllato da un singolo agente egoista e razionale che detiene come informazione privata il costo di attraversamento dell'arco che detiene. *Regole*:

- Tipo dell'agente i t_i: costo di attraversamento dell'arco
- Obiettivo del meccanismo: trovare il cammino minimo tra i nodi s e t in base ai costi reali t_i, che il sistema non conosce.
- Il meccanismo incentiva gli agenti a fornire il costo reale di attraversamento dell'arco pagando una determinata quantità p_i
- Utilità dell'agente i: $u_i = p_i - t_i$.

4.4 Progettazione di un meccanismo truthful

Poiché i problemi sono similari, per la progettazione di meccanismi truthful ci si concentrerà esclusivamente sui problemi di minimazzione, in cui:

- Per ogni outcome $x \in F$, quindi tra quelli ammissibili, la funzione di valutazione $v_i(t_i, x)$ rappresenta il costo che un player i deve sostenere in un determinato outcome x.
- La funzione di scelta sociale $f(x)$ è un problema di ottimizzazione di minimizzazione, ovvero tra tutti gli outcome ammissibili rispetto al vettore dei tipi reali t il il sistema sceglie come soluzione x quella che minimizza una determinata misura. *Esempio*: nel caso del cammino minimo, il sistema deve scegliere come soluzione il cammino che minimizza la somma dei pesi degli archi.
- Il meccanismo incentiva i player pagandoli.

Esistono due classi di meccanismi truthful, VCG e One Parameter, che sono applicabili ad una determinata categoria di problemi, chiamati **problemi utilitari**. In altre parole, se si ha un problema di ottimizzazione in uno scenario non cooperativo, se il problema è anche utilitario allora esiste un meccanismo truthful che fornisce una soluzione per il problema. Un problema è utilitario se ciò che si vuole minimizzare è la somma delle valutazioni dei singoli player i rispetto all'outcome, ovvero tra tutti gli outcome f, il sistema vuole selezionare l'outcome x che minimizza la somma dei costi sostenuti dai player in x:

$$f(t) = argmin_{x \in F} \sum_i v_i(t_i, x) \tag{4.5}$$

L'asta di Vickery è un problema utilitario. *Esempio*: il sistema vuole affidare un job, le macchine impiegano un determinato t_i, il costo in termini temporali che devono sostenere per processare il job: la funzione da minimizzare è la somma delle valutazione dei player i rispetto all'outcome. Scelto un vincitore k, $f(t) = argmin_{x \in F} \sum_i v_i(t_i, x) = t_k$ poichè le valutazione dei player $j \neq k$ a cui non viene affidato il job è pari a 0.

4.5 Meccanismo VCG

Il meccanismo VCG è l'unico meccanismo a strategie dominanti per i problemi utilitari. Il meccanismo ha in input gli r_i riportati dai player i ed è caratterizzato da:

- L'algoritmo del meccanismo $g(r)$ che calcola:

$$x = argmin_{y \in F} \sum_i v_i(r_i, y) \tag{4.6}$$

- Schema di pagamento per il player i:

$$p_i(x) = h_i(r_{-i}) - \sum_{j \neq i} v_j(r_j, x) \tag{4.7}$$

  Dove $h_i(r_{-i})$ rappresenta un valore arbitrario che non dipende da ciò che viene dichiarato dal player i-esimo, $\sum_{j \neq i} v_j(r_j, x)$ rappresenta la somma delle valutazioni dei player rispetto alla soluzione valutata escluso però il player i-esimo. *Esempio*: nel caso del cammino minimo, bisogna considerare per l'outcome selezionato le valutazioni di tutti gli agenti, e quindi gli archi selezionati, escluso l'agente (arco) i-esimo.

Teorema 11. *Il meccanismo VCG è sempre truthful per i problemi utilitari.*

Dimostrazione. Fissato un player i, le dichiarazioni dei restanti player r_{-i} e t_i. Sia $R = (r_{-i}, t_i)$ definito appunto come il vettore di ciò che non dichiarano gli altri e il tipo reale del player i e si consideri una strategia $r_i \neq t_i$, ovvero il player i non dichiara il suo tipo reale ma il falso, sovradichirando o sottodichiarando.
1) Se il player i dichiara il suo tipo reale, quindi $r_i = t_i$, allora l'algoritmo calcolerà la soluzione:

$$x = g(r_{-i}, t_i) = g(R)$$

che dipende dal vettore dei tipi.
2) Se il player i mente, quindi $r_i \neq t_i$, allora l'algoritmo calcolerà una soluzione $x' \neq x$:

$$x' = g(r_{-i}, r_i)$$

Osserviamo ora le utilità in entrambi i casi 1) e 2).
Caso 1

$$u_i(t_i, x) = [h_i(r_{-i}) - \sum_{j \neq i} v_j(r_j, x)] - v_i(t_i, x) = h_i(r_{-i}) - \sum_j v_j(R_j, x) \tag{4.8}$$

Caso 2

$$u_i(t_i, x') = [h_i(r_{-i}) - \sum_{j \neq i} v_j(r_j, x')] - v_i(t_i, x') = h_i(r_{-i}) - \sum_j v_j(R_j, x') \tag{4.9}$$

Concentrandoci sul primo caso (il secondo è simile), inserendo all'interno della sommatoria il termine $v_i(t_i, x)$ si ottiene la somma delle valutazioni di tutti i player rispetto al seguente vettore dei tipi: tutti i player j hanno dichiarato r_j, l'i-esimo player ha dichiarato il tipo reale t_i. Si tratta, come si può notare,

proprio del vettore R (esempio per il problema dei cammini minimi: la sommatoria rappresenta il costo del cammino x se si pesano gli archi secondo il vettore R).
Ricordiamo che l'algoritmo del meccanismo restituirà come output la soluzione migliore rispetto ai tipi riportati. Poiché l'algoritmo del meccanismo trova la soluzione migliore rispetto alle dichiarazioni dei player, al player i conviene dichiarare sempre $r_i = t_i$ dal momento che:

$$\sum_j v_j(R_j, x) \leq \sum_j v_j(R_j, x') \tag{4.10}$$

$$u_i(t_i, x) \geq u_i(t_i, x') \tag{4.11}$$

□

4.5.1 Come definire $h_i(r_{-i})$

Il pagamento deve essere necessariamente maggiore rispetto al costo sostenuto dal player, poiché nello schema dei pagamenti definito compaiono solo valori negativi e il player deve essere incentivato a giocare. Non è un caso che nella formula del pagamento compaia anche $h_i(r_{-i})$ $(\neq 0)$ che deve essere scelto grande a piacere per fare in modo che $p_i(x) > 0$.
I pagamenti di Clarke
$h_i(r_{-i})$ è la soluzione che l'algoritmo calcolerebbe quando il player i non partecipa al gioco (la seconda migliore soluzione senza il player i):

$$h_i(r_{-i}) = \sum_{j \neq i} v_j(r_j, g(r_{-i})) \tag{4.12}$$

$$p_i(x) = \sum_{j \neq i} v_j(r_j, g(r_{-i})) - \sum_{j \neq i} v_j(r_j, g(r)) \tag{4.13}$$

I pagamenti di Clarke: Vickery Auction Min

$$x = g(r) = argmin_{x \in F} \sum_i v_i(r_i, x) \tag{4.14}$$

Il job viene affidato alla macchina che riporta il costo più basso. Supponiamo che vinca il player i:

$$p_i(x) = \sum_{j \neq i} v_j(r_j, g(r_{-i})) - \sum_{j \neq i} v_j(r_j, g(r)) \tag{4.15}$$

Chi non vince il job viene quindi pagato 0 come si evince chiaramente dalla formula.

4.5.2 Complessità meccanismo

La complessità del meccanismo consiste nella complessità dell'algoritmo $g(r)$, ovvero quanto impiega a calcolare la soluzione dati i tipi riportati dai player, e la complessità per calcolare la funzione di pagamento.

4.6 Shortest Path in un grafo con archi privati

Richiami: Un problema si definisce utilitario quando il sistema viene progettato per trovare, tra tutte le soluzioni ammissibili, quella che minimizza (massimizza) la somma delle valutazioni di tutti i player. Per i problemi utilitari esistono dei meccanismi veritieri: i meccanismi VCG.
Tra i problemi utilitari trattati nel capitolo precedente vi è anche il problema della ricerca del cammino minimo in un grafo $G(V, E)$ in cui gli archi e sono detenuti da agenti egoisti e razionali, che privatamente conoscono il costo (o peso) di attraversamento dell'arco. Riassumento:

- Input del problema: grafo $G(V, E)$.
- Il nodo s è il nodo sorgete, il nodo t è il nodo destinazione.
- Gli archi $e \in E$ sono controllati da agenti egoisti.
- Solo l'agente conosce il peso dell'arco e che detiene, che corrisponde al costo di utilizzo dell'arco ($tipo > 0$), e inoltre la valutazione $v = tipo$.
- L'obiettivo è calcolare una buona soluzione di un certo problema di ottimizzazione rispetto ai pesi reali, ossia calcolare il vero cammino minimo in $G(V, E, tipi)$ tra s e t tra tutti i cammini minimi tra s e t.
- Si progetta un meccanismo truthful, in cui il meccanismo paga opportunatamente gli agenti per convincerli a rivelare i pesi reali degli archi.
- Il meccanismo può decidere, date le dichiarazioni degli agenti, il cammino e lo schema di pagamento.

Entrando nel dettaglio, e spiegando più formalmente lo scenario:

- Soluzioni ammissibili F: insieme di tutti i possibili cammini in G da s a t.
- Tipo dell'agente e: t_e, il peso dell'arco che rappresenta il costo sostenuto dall'agente per l'uso dell'arco.
- Valutazione dell'agente e di un cammino qualsiasi $P \in F$: $v_e(t_e, P) = t_e$ se $e \in P$, 0 altrimenti.
- Utilità dell'agente e: $u_e = p_e - t_e$

4.6.1 Progettazione del meccanismo truthful

L'osservazione cruciale per questo problema è dimostrare che il problema è utilitario. Se si prende una soluzione P del problema e si effettua la somma, su tutti gli archi del grafo, della valutazione dei player, rispetto ai tip associati, si ottiene che se $e \notin P$ allora $v_e = 0$, altrimenti $v_e = t_e$. Tale somma altro non è che la lunghezza vera (pesata) del cammino P:

$$\sum_{e \in E} v_e(t_e, P) \tag{4.16}$$

La lunghezza di un cammino è la somma delle valutazioni di tutti gli agenti, si vuole trovare il cammino che minimizza la lunghezza, cioè tra tutte le soluzioni si vuole trovare quella che minimizza la somma delle valutazioni di tutti gli agenti. Quindi il problema è utilitario e per tali problemi si usano i meccanismi VCG.

Meccanismo VCG $M =< g(r), p(x) >$:

- l'algoritmo $g(r)$ ritorna come soluzione, rispetto ai tipi riportati r_j:

$$x^* = argmin_{x \in F} \sum_j v_j(r_j, x) \tag{4.17}$$

- per lo schema di pagamento $p_e(x)$ viene calcolato, per ogni arco $e \in E$:

$$p_e = \sum_{j \neq e} v_j(r_j, g(r_{-e})) - \sum_{j \neq e} v_j(r_j, x^*) \tag{4.18}$$

Si noti che il termine sottrattivo $\sum_{j \neq e} v_j(r_j, x^*)$ è la somma delle valutazioni di tutti gli agenti escluso l'agente e che il sistema sta pagando rispetto alla soluzione che il sistema ha calcolato. Il termine $\sum_{j \neq e} v_j(r_j, g(r_{-e}))$, che rispetto al player e è una costante poiché non dipende da esso, è definito come la somma delle valutazioni di tutti i player escluso il player e rispetto ad una determinata soluzione x calcolata quando il player e non partecipa al gioco.

Meccanismo VCG per lo Shortest Path $M_{SP} =< g(r), p(x) >$:

- l'algoritmo $g(r)$ calcola il cammino minimo $P_G(s,t)$ in $G(V, E, r)$ (ovvero in base ai tipi riportati)
- per lo schema di pagamento $p_e(x)$ viene calcolato, per ogni arco $e \in E$:

$$p_e(P_G(s,t)) = \sum_{j \neq e} v_j(r_j, g(r_{-e})) - \sum_{j \neq e} v_j(r_j, P_G(s,t)) \tag{4.19}$$

  Ovvero, $p_e =$
  - se $e \in E$:

$$d_{G-e}(s,t) - (d_G(s,t) - r_e) \tag{4.20}$$

  - altrimenti 0.

Quindi per gli archi che non sono stati selezionati il player non guadagna nulla, ma d'altronde non sostiene alcun costo, per gli archi selezionati il pagamento è stato descritto sopra. Si noti che $(d_G(s,t) - r_e)$ altro non è che il peso del cammino senza il peso dell'arco e che si sta considerando, mentre $d_{G-e}(s,t)$ è il cammino minimo nel grafo senza l'arco e anche chiamato cammino minimo di rimpiazzo in $G - e = (V, E - e, r_{-e})$ tra i nodi s e t.

Esempio
Sia G(V, E) il grafo pesato rappresentato in figura **??**.

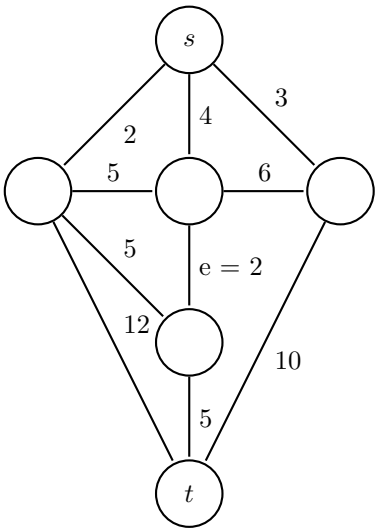


Figura 4.1: G(V, E)

Il cammino minimo $P_G(s,t)$ è rappresentato in figura **??**.

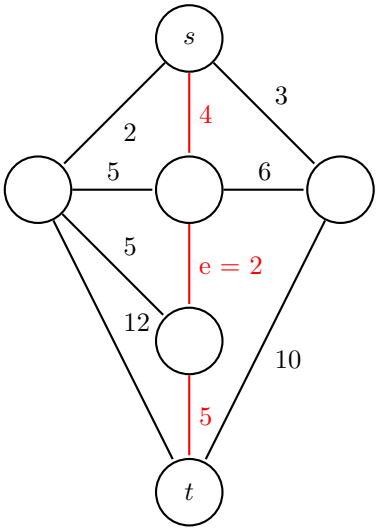


Figura 4.2: G(V, E)

Concentriamoci sull'arco e, il suo pagamento dipenderà dal cammino minimo di rimpiazzo $P_{G-e}(s,t)$, rappresentato in figura **??**.

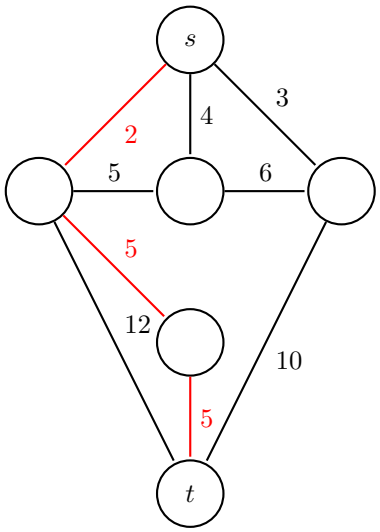


Figura 4.3: G(V, E)

L'arco e viene quindi pagato:

p_e = lunghezza del cammino di rimpiazzo−lunghezza del cammino minimo escluso l'arco e =

$$12 - (11 - 2) = 3$$

Il risultato ottenuto rappresenta di fatto la dichiarazione massima che l'agente e può fare per essere selezionato. Se l'agente sottodichiara, allora il cammino minimo continua ad essere quello rappresentato in **??**. Se l'agente sovradichiara, allora il cammino minimo è rappresentato in figura **??**. Il sistema, dunque, paga all'arco un contributo marginale.
Questo esempio già fa intuire il motivo per cui ad un agente conviene dichiarare la verità. L'agente, non conoscendo le dichiarazioni degli altri player, non può infatti sapere la sua soglia di uscita: sicuramente non può rischiare di sottodichiarare poiché se la soglia di uscita è più bassa di quanto sottodichiara rischierebbe di avere un utile negativo. Supponiamo infatti che la soglia di uscita sia 1, il tipo reale dell'agente è $t_e = 2$ ma, pur di essere selezionato, l'agente dichiara sotto la soglia. E' chiaro che l'agente sarebbe selezionato ma avrebbe un pagamento di 1 e un costo di 2, e l'utile sarebbe negativo.

4.6.2 Complessità

Il meccanismo deve sostanzialmete calcolare la soluzione e i pagamenti secondo lo schema definito. La complessità del meccanismo è quindi la complessità necessaria per calcolare la soluzione (shortest path tra due nodi) più la complessità necessaria per calcolare i pagamenti. Si tratta, alla fine, solo di una questione algoritmica: più viene risolta efficientemente, più si ottiene un meccanismo veloce.

Notazione

- $n = |V|$, $m = |E|$

- $d_G(s,t)$: distanza in G da s a t, cioè la somma dei pesi degli archi di $P_G(s,t)$
- Assunzione: s e t sono 2-edge connessi ovvero se dal grafo G si elimina uno e un solo arco allora nel grafo G' esiste un altro cammino tra s e t. In altre parole, per ogni arco e esiste sempre un cammino minimo alternativo in $G - e$.
- Se la proprietà 2-edge connessione non è più garantita, e quindi esiste un arco ponte e che disconnette il grafo in due componenti C_1 e C_2, e $s \in C_1$ e $t \in C_2$, allora tutti i cammini alternativi tra s e t passano per l'arco e.
- Se l'arco e è un ponte allora $d_{G-e}(s,t) = \infty$. Il player che detiene quell'arco tiene in pugno il sistema e può quindi chiedere qualsiasi cifra.

Una soluzione
- Per il calcolo del cammino minimo: Algoritmo di Dijkstra. **Complessità**: $O(m + nlogn)$
- Per il calcolo dei pagamenti, $\forall e \in P_G(s,t)$ si applica l'algoritmo di Dijkstra al grafo $G - e$. **Complessità**: $O(mn + n^2logn)$ poichè vi sono $k = O(n)$ archi.

Teorema 12. *M_{SP} è calcolabile in tempo $O(m + nlogn)$*

4.7 Limiti del VCG

I meccanismi VCG possono essere utilizzati quando il problema è utilitario, e un problema è utiltario quando la funzione obiettivo del sistema è la seguente: tra tutte le soluzioni ammissibili, viene selezionata quella che minimizza la somma delle valutazioni.
Nello scenario seguente, invece, il meccanismo VCG non può essere applicato.

Shortest Path Tree non cooperativo: broadcasting
Un nodo s vuole spedire un messaggio a tutti i nodi x della rete, $x \in V - s$. L'informazione posseduta dai singoli agenti è il tempo di attraversamento degli archi, ovvero il tempo impiegato da un messaggio per attraversare un arco. Si immagini di spedire i messaggi lungo dei cammini in modo tale che la consegna avvenga il prima possibile, ad esempio tramite un albero. Si sta dunque calcolando uno Shortest Path Tree del grafo. L'SPT è è un albero T radicato in s con la seguente proprietà:

Osserviamo il cammino nell'albero T tra il nodo s e un qualsiasi altro nodo x: la lunghezza del cammino nell'albero è esattamente la distanza (il cammino minimo) nel grafo G.

Ne consegue che la funzione obiettivo che si sta ottimizzando è la seguente. Sia F l'insieme degli alberi ricoprenti di V radicati nel nodo s. $\forall T \in F$:

$$f(t) = argmin_{T \in F} \sum_{v \in V} d_T(s,v) = argmin_{T \in F} \sum_{e \in E(T)} t_e ||e|| \tag{4.21}$$

Ogni arco e ha un peso che dipende dalla sua molteplicità, intesa come numero di cammini ai quali appartiene in T.
Si veda l'esempio in figura **??** di un albero ricoprente del grafo: l'arco (s, b) di costo 1 viene attraversato il numero di nodi che fanno parte del sottoalbero. La molteplicità $||e|| = 5$

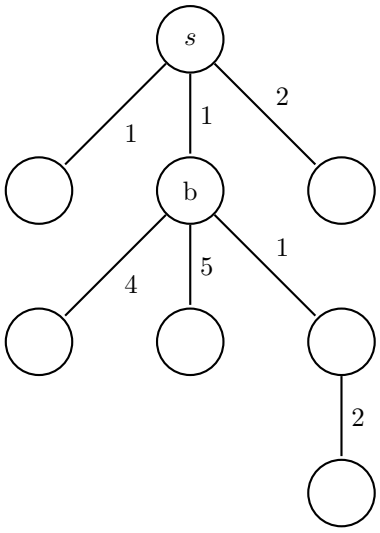


Figura 4.4: T

Il problema è utilitario? Dipende fortemente dalla tipologia delle valutazioni. Se si utilizzasse come protocollo di broadcast il protocollo unicast: il nodo s invia un messaggio distinto per ogni nodo, e il costo che sostiene l'agente dell'arco e è $t_e * \#messaggi$, ovvero $t_e * ||e||$ allora si tratta di un problema utilitario.
Se si utilizzasse il protocollo multicast, in cui viene inviata una sola copia del messaggio su ogni arco (il messaggio viene duplicato in ogni nodo), e t_e è il costo che sostiene l'arco per far attraversare il messaggio, su ogni arco transita un solo messaggio ma allora $v_e(t_e, T) = t_e$ se $e \in E(T)$, altrimenti 0 e inoltre:

$$f(t) \neq argmin_{T \in F} \sum_{e \in E(T)} v_e(t_e, T) \tag{4.22}$$

Ne consegue che il problema non è utilitario e il meccanismo VCG non è truthful.

4.8 One Parameter

Nel caso in cui i problemi non siano utilitari, quali meccanismi è possibile utilizzare? Per i problemi One Parameter si usano i meccanismi One Parameter che, a differenza dei meccanismi VCG, hanno un sapore più algoritmico, poiché la caratterizzazione di questi meccanismi ha a che fare con una proprietà specifica dell'algoritmo del meccanismo.
Un problema di Mechanism Design è One Parameter se valgono le seguenti **condizioni**:

- L'informazione posseduta da ogni agente a_i (quindi il suo tipo) è un singolo parametro reale: $t_i \in R$

- La valutazione di a_i deve avere la forma:

$$v_i(t_i, o) = t_i w_i(o) \tag{4.23}$$

dove $w_i(o)$ è il carico di lavoro per l'agente a_i sulla soluzione o.

Per tutti i problemi One Parameter esiste una classe ben caratterizzata di meccanismi che risolve tali problemi chiamati meccanismi One Parameter.

4.8.1 One Parameter e SPT non cooperativo

L'SPT è è un albero T radicato in s con la seguente proprietà:

Osserviamo il cammino nell'albero T tra il nodo s e un qualsiasi altro nodo x: la lunghezza del cammino nell'albero è esattamente la distanza (il cammino minimo) nel grafo G.

Ne consegue che la funzione obiettivo che si sta ottimizzando è la seguente. Sia F l'insieme degli alberi ricoprenti di V radicati nel nodo s. $\forall T \in F$:

$$f(t) = argmin_{T \in F} \sum_{v \in V} d_T(s, v) = argmin_{T \in F} \sum_{e \in E(T)} t_e ||e|| \tag{4.24}$$

$$v_e(t_e, T) = t_e \text{ se } e \in E(T), 0 \text{ altrimenti} \tag{4.25}$$

Il multicast è un problema non utilitario, come già mostrato, ma è un problema One Parameter perchè:

$$\begin{aligned} v_e(t_e, T) &= t_e w_e(T) \\ w_e(T) &= 1 \text{ oppure } w_e(T) = 0 \end{aligned} \tag{4.26}$$

4.8.2 Monotonia

Un algoritmo $g()$ che calcola una soluzione (la soluzione determina anche il carico di lavoro per ogni singolo agente) per un problema One Parameter di minimizzazione è monotono se per ogni agente a_i, $w_i(g(r_{-i}, r_i))$ è non crescente rispetto a r_i, per tutti gli $r_{-i} = (r_1, ..., r_{i-1}, r_{i+1}, ..., r_N)$. In altre parole un algoritmo è monotono se per ogni agente a_i, per ogni possibile configurazione dei tipi degli altri agenti, il carico di lavoro che il sistema assegna ad a_i rispetto a ciò che lui dichiara deve essere non crescente, e quindi più lui dichiara (più il suo costo è alto) più l'algoritmo deve assegnare un carico di lavoro basso.
Notazione per semplificare: $w_i(g(r)) = w_i(r)$, $p_i(g(r)) = p_i(r)$.

Teorema 13. *Condizione necessaria affinché un meccanismo $M =< g(r), p(r) >$ per un problema One Parameter sia veritiero è che $g(r)$ sia monotono.*

Dimostrazione. Per assurdo.
Supponiamo che $g()$ non sia monotono, si farà vedere che se $g()$ non è monotono il meccanismo M non sarà mai truthful. Se $g()$ non è monotono allora esiste un agente a_i e un vettore di strategie r_{-i} tale che $w_i(r_{-i}, r_i)$ è non non crescente (attenzione: non vuol dire crescente).
Per la dimostrazione si farà riferimento alla figura **??**, dove r_{-i} è fissato, sull'asse

delle ordinate è presente il carico di lavoro che l'algoritmo assegna ad un input in funzione di r_i mentre sull'asse delle ascisse è presente ciò che dichiara l'agente, ovvero r_i.

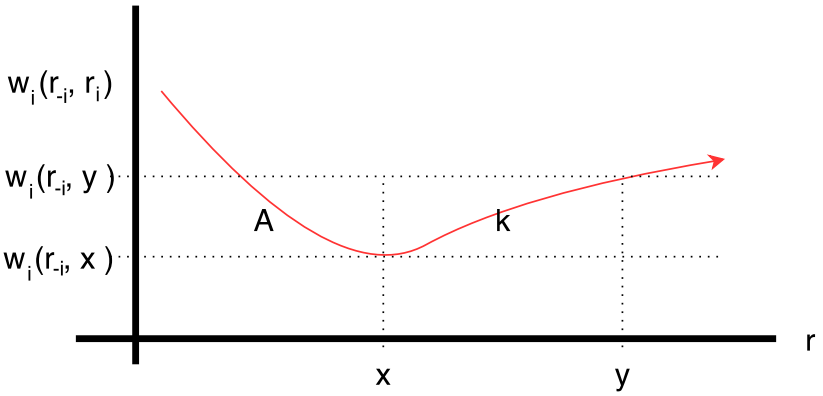


Figura 4.5: Non crescente

Confrontiamo ora le dichiarazioni e il carico di lavoro calcolato.
Caso 1: $t_i = x$ e $r_i = t_i \rightarrow v_i(t_i, o) = x * w_i(r_{-i}, x)$

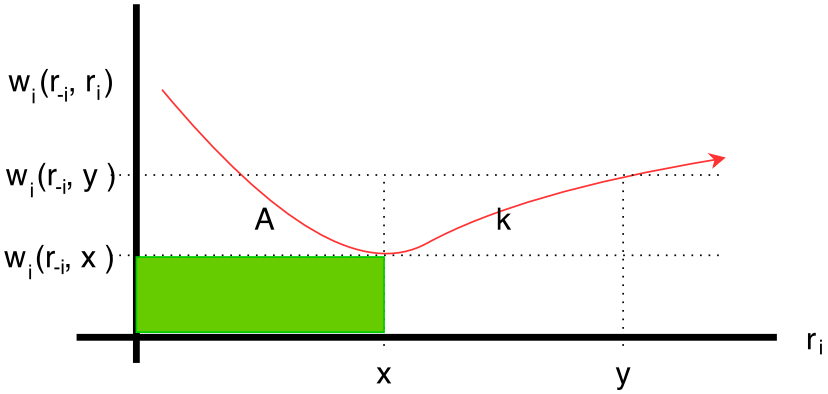


Figura 4.6: Caso 1

Caso 2: $t_i = y$ e $r_i = t_i \rightarrow v_i(t_i, o) = x * w_i(r_{-i}, y)$

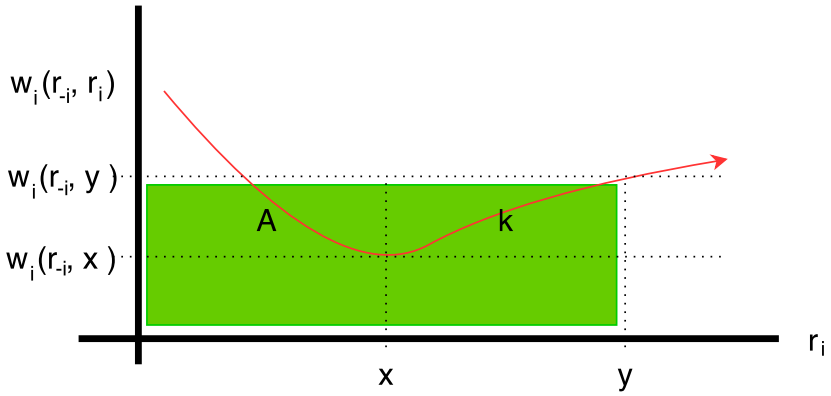


Figura 4.7: Caso 2

Caso 3: $t_i = x$ e $r_i = y \rightarrow a_i$ aumenta il suo costo di A

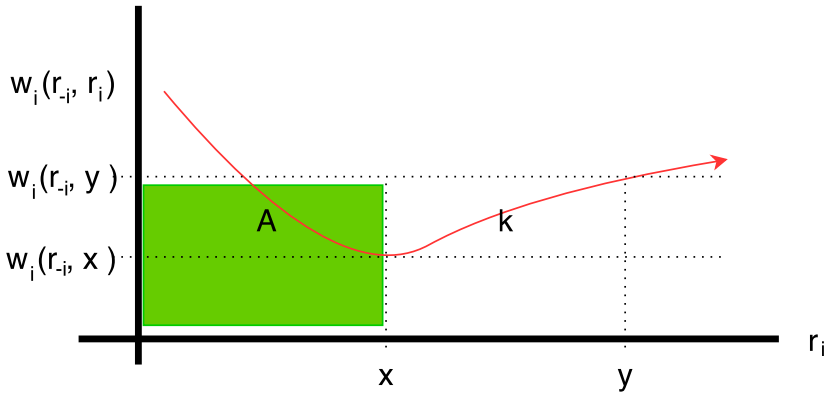


Figura 4.8: Caso 3

Caso 4: $t_i = y$ e $r_i = x \rightarrow a_i$ ha un risparmio di $A + k$

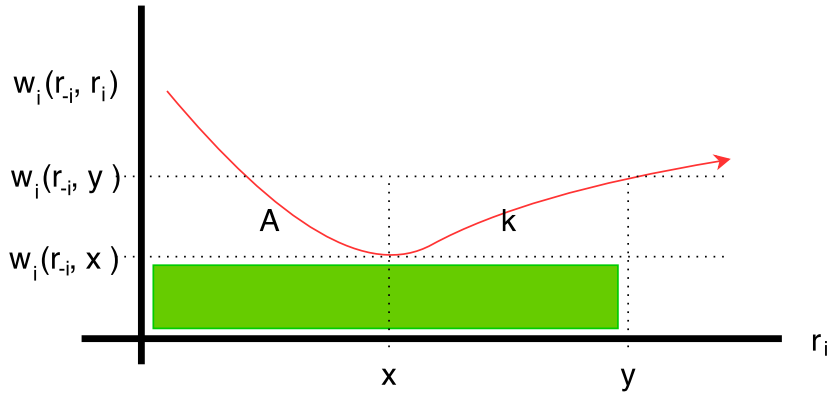


Figura 4.9: Caso 4

Sia $\Delta p = p_i(r_{-i}, y) - p_i(r_{-i}, x)$ la differenza di pagamento quando l'agente dichiara x e quando dichiara y. Se il meccanismo M è truthful deve essere:

- La differenza di pagamento non deve essere maggiore di A: $\Delta p \leq A$. Questo perchè se $t_i = x$, a_i potrebbe dichiarare il falso ovvero y e avrebbe un incremento di costo pari ad A, e quindi se $\Delta p > A$ avrebbe un incremento di pagamento pari a Δp ovvero la sua utilità aumenta.
- La differenza di pagamento non deve essere minore di $A + k$, altrimenti quando $t_i = y$, a_i potrebbe dichiarare il falso ovvero x in quanto in tal caso il suo costo diminuisce di $A + k$, e quindi se $\Delta p < A + k$ ciò significa che il decremento nel pagamento è minore del decremento del costo ovvero la sua utilità aumenta.

Se non fosse che k è strettamente positivo, e Δp deve essere contemporaneamente minore di A e maggiore di una quantità più grande di $A + k$.

□

4.8.3 Schema di pagamento

Sia $g(r)$ un qualsiasi algoritmo monotono, e sia $p_i(r)$ lo schema di pagamento seguente:

$$p_i(r) = h_i(r_{-i}) + r_i w_i(r) - \int_0^{r_i} w_i(r_{-i}, z)dz \tag{4.27}$$

dove $h_i(r_{-i})$ è una funzione arbitraria indipendente da r_i, e r_i le dichiarazioni degli altri agenti. In sostanza, oltre a questa costante, nella formula è presente, se l'agente ha dichiarato r_i e l'algoritmo ha calcolato un carico di lavoro pari a $w_i(r)$, anche il costo sostenuto dall'agente meno l'integrale del carico di lavoro.

Teorema 14. *Un meccanismo One Parameter è veritiero, se si utilizza lo schema di pagamento illustrato.*

Dimostrazione. Faciamo vedere che l'utilità di un agente a_i può solo decrescere se a_i mente. Poniamo, nella formula del pagamento, $h_i(r_{-i}) = 0$ perchè è una funzione costante che non dipende da ciò che dichiara l'agente a_i. L'utilità dell'agente a_i è pari a:

Caso 1 a_i dichiara $r_i = t_i$

$$\begin{aligned} &u_i(t_i, g(r_{-i}, t_i)) = p_i(g(r_{-i}, t_i)) - v_i(t_i, g(r_{-i}, t_i)) = \\ &t_i w_i(g(r_{-i}, t_i)) - \int_0^{t_i} w_i(r_{-i}, z)dz - t_i w_i(g(r_{-i}, t_i)) = \\ &- \int_0^{t_i} w_i(r_{-i}, z)dz \end{aligned} \tag{4.28}$$

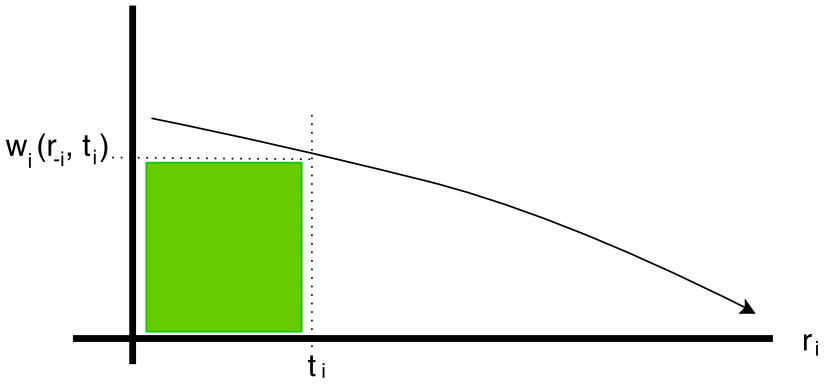


Figura 4.10: Caso 1

Caso 2 a_i dichiara $r_i = x > t_i$
Il costo diventa $C = t_i w_i(r_{-i}, x)$ mentre il pagamento è pari a:

$$P = x w_i(r_{-i}, x) - \int_0^{x} w_i(r_{-i}, z)dz \tag{4.29}$$

Ne consegue che a_i sta perdendo G.

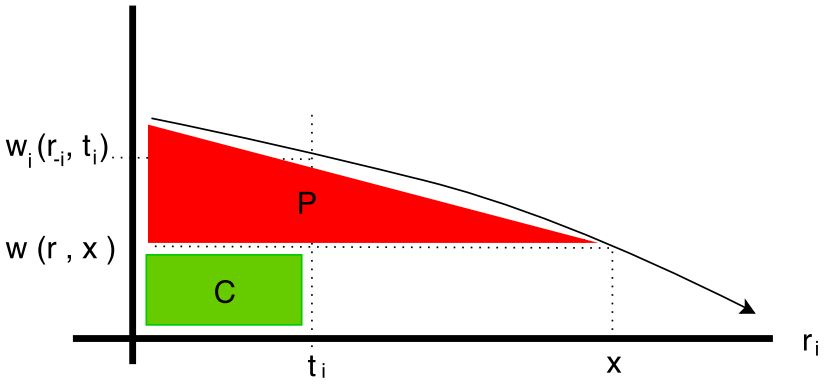


Figura 4.11: Caso 2

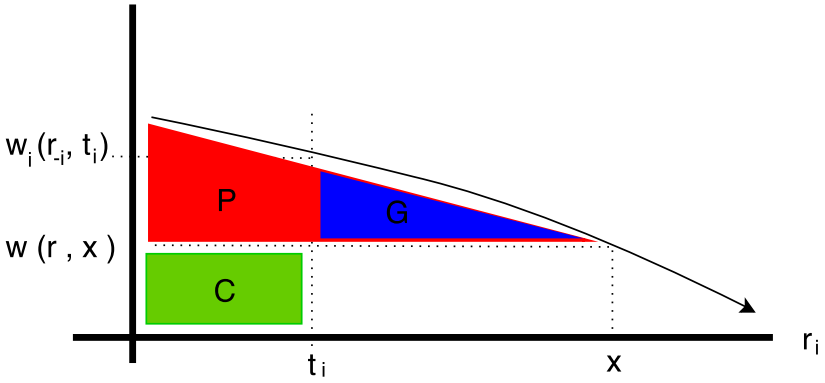


Figura 4.12: Caso 2

Caso 3 a_i dichiara $r_i = x < t_i$
Il costo diventa $C = t_i w_i(r_{-1}, x)$ mentre il pagamento è pari a:

$$P = x w_i(r_{-i}, x) - \int_0^x w_i(r_{-i}, z) dz \tag{4.30}$$

Ne consegue che a_i sta perdendo G.

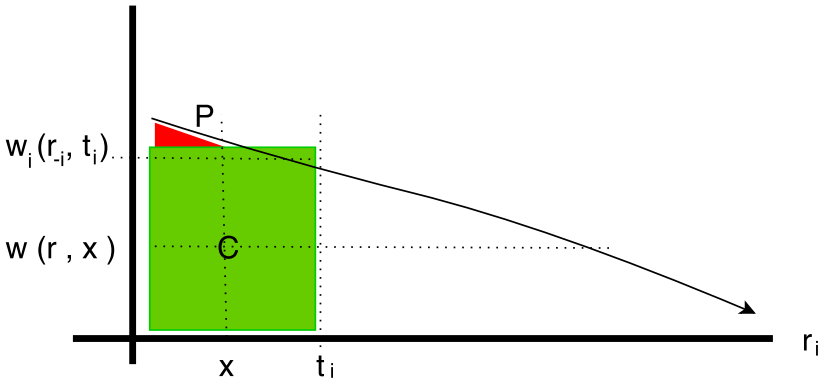


Figura 4.13: Caso 3

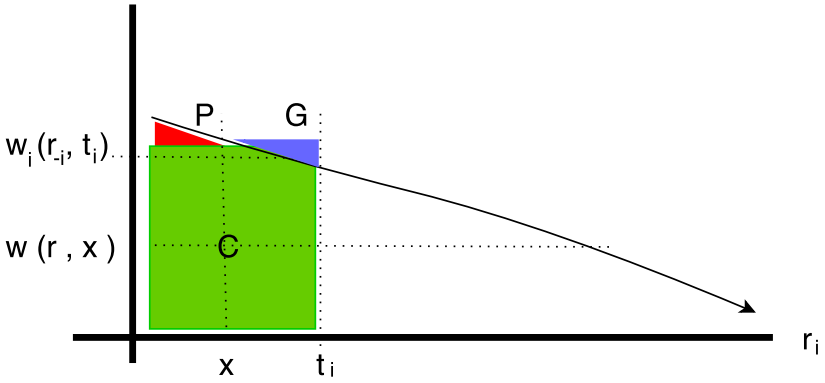


Figura 4.14: Caso 3

Conclusione a_i non ha alcuna convenienza a mentire

□

4.8.4 Come definire $h_i(r_{-i})$

Un meccanismo garantisce la volontaria partecipazione se l'utilità di un qualsiasi agente (che dichiara il vero) ha sempre un utile non negativo.
Per rendere l'utilità positiva sempre, ricordiamo prima che il pagamento dell'agente a_i quanto dichiara r_i è pari a:

$$p_i(r) = h_i(r_{-i}) + r_i w_i(r) - \int_0^{r_i} w_i(r_{-i}, z)dz \tag{4.31}$$

Il player non sta mentendo e, come si evince chiaramente, il secondo termine della formula del pagamento si annulla quando si calcola l'utilità, quindi il pagamento diventa:

$$p_i(r) = h_i(r_{-i}) - \int_0^{r_i} w_i(r_{-i}, z)dz \tag{4.32}$$

Se si sceglie:

$$h_i(r_{-i}) = \int_0^{\infty} w_i(r_{-i}, z)dz \tag{4.33}$$

E il pagamento diventa quindi:

$$p_i(r) = r_i w_i(r) + \int_{r_i}^{\infty} w_i(r_{-i}, z)dz \tag{4.34}$$

L'utilità di un agente è invece, anche per via delle considerazioni già fatte:

$$u_i(t_i, g(r)) = + \int_{r_i}^{\infty} w_i(r_{-i}, z)dz \geq 0 \tag{4.35}$$

4.8.5 Meccanismo One Parameter per Shortest Path Tree

Come già visto, il problema dello Shortest Path Tree con Multicast è un problema One Parameter perchè:

$$\begin{aligned} v_e(t_e, T) &= t_e w_e(T) \\ w_e(T) = 1 \text{ oppure } w_e(T) &= 0 \end{aligned} \tag{4.36}$$

Si vuole progettare il meccanismo $M_{SPT} =< g(r), p(r) >$ che consiste di:

- $g(r)$: dato il grafo e le dichiarazioni, calcola un SPT $S_G(s)$ di $G = (V, E, r)$ utilizzando l'algoritmo di Dijkstra.
- $p(r)$: per ogni arco $e \in E$:

$$p_e(r) = r_e w_e(r) + \int_{r_e}^{\infty} w_e(r_{-e}, z)dz \tag{4.37}$$

  per garantire la partecipazione volontaria

Il carico di lavoro di un agente a_e è 1 se l'arco viene selezionato, 0 altrimenti. Fissato un arco e e i costi degli archi tranne l'arco e, quindi r_e, si osservi come cambia la soluzione, lo SPT del grafo, se si varia il costo dell'arco e.
Se si pesa l'arco e 0, calcolato lo SPT l'arco viene quasi sicuramente selezionato e quindi il carico è 1. Se si pesa l'arco e con un valore elevato, calcolato lo SPT l'arco esce molto probabilmente dalla soluzione, e - una volta uscito - se il peso dell'arco e viene ancora aumentato, l'arco a maggior ragione non farà parte della soluzione. Si sta dicendo che, osservando ogni singolo arco, la funzione carico di lavoro sarà sempre una funzione a gradoni in cui c'è una soglia Θ_e, che in realtà è una soglia che dipende soltanto da r_{-e}, tale che se l'arco viene pesato un valore strettamente più piccolo della soglia, allora l'arco viene selezionato nella

soluzione, altrimenti l'arco non è selezionato nello Shortest Path Tree. Questa soglia può essere anche 0 e ∞ (ad esempio se si tratta di un arco ponte del grafo).

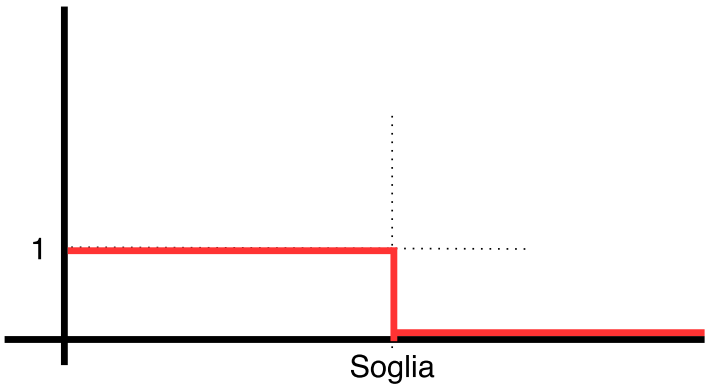


Figura 4.15: Carico di lavoro a gradoni

Definiamo i **pagamenti**:
$p_e = 0$ se e non è un arco selezionato:

$$p_e(r) = r_e w_e(r) + \int_{r_e}^{\infty} w_e(r_{-e}, z)dz = 0 + 0 = 0 \tag{4.38}$$

$p_e = \Theta_e$ se e è nella soluzione:

$$p_e(r) = r_e w_e(r) + \int_{r_e}^{\infty} w_e(r_{-e}, z)dz = r_e + \Theta_e - r_e = \Theta_e \tag{4.39}$$

Ovvero, in questo secondo caso, si tratta del massimo valore che l'arco può dichiarare per essere selezionato.

4.8.6 Binary Demand

Un problema è Binary Demand, sottoclasse dei problemi One Parameter, se il carico di lavoro è 0 o 1 (a gradoni) e l'algoritmo è monotono.

Esempio
Sia $e = (u, v)$ un arco in $S_G(s)$ (figura **??**, e vediamo quale range di valori può dichiarare e per essere selezionato.

Caso $r_e = 1$ l'arco fa parte della soluzione.

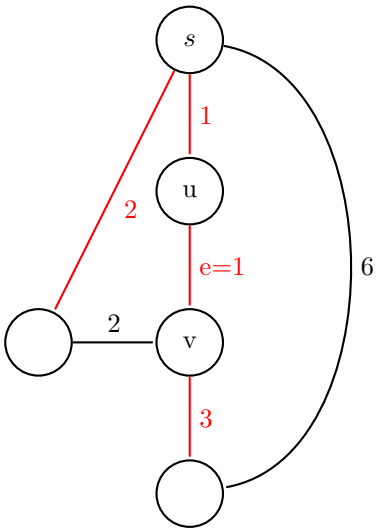


Figura 4.16: Esempio

Caso $r_e = 2 + \epsilon$ l'arco fa parte della soluzione

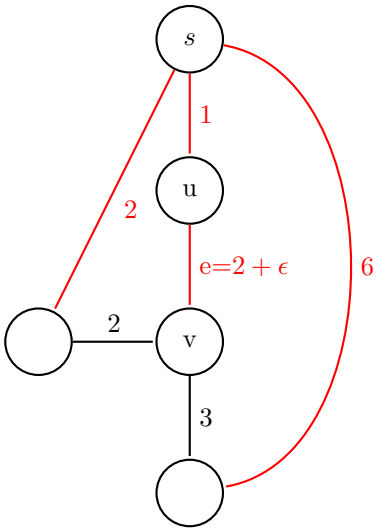


Figura 4.17: Esempio

Caso $r_e = 3 + \epsilon$ l'arco non fa parte della soluzione.

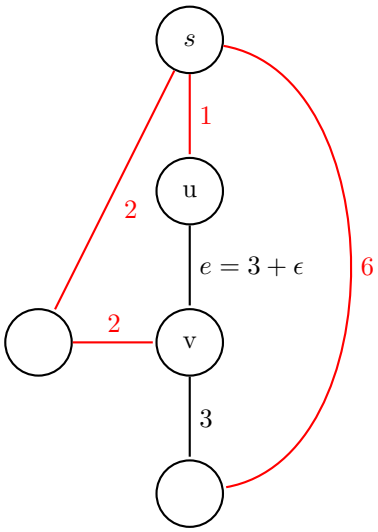


Figura 4.18: Esempio

L'arco (u, v) esce dalla soluzione quando supera $r_e = 3$ perchè l'arco serve per arrivare al nodo v. Quando il cammino supera il costo 4 del cammino alternativo per arrivare al nodo v, allora l'arco esce dalla soluzione.

Allora la soglia di uscita è pari a:

$$\Theta_e = d_{G-e}(s, v) - d_G(s, u) \tag{4.40}$$

4.8.7 Complessità

Il meccanismo deve sostanzialmete calcolare la soluzione e i pagamenti secondo lo schema definito. Per quanto riguarda i pagamenti si applica, per ogni arco $e = (u, v) \in S_G(s)$, l'algoritmo di Dijkstra al grafo $G - e$ e si trova $d_{G-e}(s, v)$. Ne consegue che, essendo gli archi $k = n - 1$, la complessità è pari a: $O(mn + n^2 logn)$.

Teorema 15. *M_{SPT} è calcolabile in tempo $O(m + nlogn)$*

4.9 Aste combinatoriche

4.9.1 Richiami

Un problema di Mechanism Design è un problema di ottimizzazione in cui c'è il sistema che deve trovare una buona soluzione per il problema di ottimizzazione, in cui però alcuni input - come gli archi di un grafo, o il valore degli oggetti - sono ignoti al sistema, poichè è una informazione privata detenuta da agenti egoisti. Il sistema può chiedere di fare una dichiarazione per rivelare l'informazione privata ma gli agenti possono mentire. L'obiettivo del sistema è progettare un meccanismo, ovvero un algoritmo con uno schema di pagamento che combinati assieme garantiscono che gli agenti cooperano con il sistema

rivelando la verità, e il sistema usando l'algoritmo del meccanismo restituisce come output una soluzione per il problema di ottimizzazione. Il meccanismo viene detto truthful se l'algoritmo e lo schema di pagamento garantiscono che è strategia dominante per i player rivelare la verità.
L'obiettivo dei player è ottimizzare una funzione di utilità che:

- Nei problemi di minimizzazione, $u_i = p_i - r_i$
- Nei problemi di massimizzazione, $u_i = r_i - p_i$

Tra gli esempi visti, ricordiamo l'asta per la vendita di un singolo oggetto. Ogni player i ha come informazione privata il tipo, ovvero il valore reale dell'oggetto che si identifica con t_i. Essendo un'asta sociale, il sistema vuole vendere l'oggetto al player che lo valuta di più. Ovvero si sta cercando il massimo tra tutti i t_i, anche se tali valori sono ignoti. Stabilite le regole dell'asta, gli agenti faranno una valutazione r_i che potenzialmente differisce dal valore reale dell'oggetto e la consegneranno in busta chiusa. In base alle regole dell'asta, uno dei player i vince l'oggetto, in base allo schema di pagamento il player vincitore pagherà l'oggetto una determinata cifra, l'utilità di un player è:

- Se il player non vince: 0
- Se il player vince: $u_i = t_i - p$

Lo scopo del sistema è di vendere l'oggetto a chi lo valuta maggiormente, e tale scopo si raggiunge solo con un meccanismo veritiero. Come già visto, il meccanismo veritiero per questa tipologia di problemi (second price option) prevede che:

- Vince il player che fa l'offerta maggiore
- Il player vincitore paga la seconda migliore offerta

Con queste regole, ad ogni player conviene dichiarare la verità.

L'asta Second Price Option è un caso specifico della classe di problemi VCG, che si applicano a tutti i problemi utilitari mentre, per quelli non utiltari, a volte è possibile applicare il meccanismo One Parameter.

4.9.2 Scenario

L'asta combinatorica è una generalizzazione dell'asta a singolo oggetto. Caratteristiche:

- Il sistema non vende più un singolo oggetto, ma un insieme di oggetti indivisibili
- Partecipano all'asta n player, ogni player è interessato ad un sottoinsieme degli oggetti.
- L'informazione privata detenuta dai player: valutazione reale dell'intero insieme.

Esempio
Vengono messe all'asta una pizza, una birra, un pacchetto di patatine fritte, una fetta di torta.

1. $t_1 = 20$, interessato a Birra, Pizza e Patatine
2. $t_2 = 15$, interessato a Pizza e Torta
3. $t_3 = 6$, interessato a Birra

Dove t_i denota la contentezza del player rispetto agli oggetti di cui è interessato. Ovviamente, ad esempio, nel caso in cui al player 3 venga data oltre alla Birra anche la Torta, il suo valore di contetezza t_i rimane invariato.

Essendo l'asta sociale, il sistema vuole decidere chi sono i vincitori, come allocare gli oggetti e quanto ogni singolo vincitore dovrà pagare il suo sottoinsieme di oggetti. Informalmente l'insieme delle soluzioni ammissibili F sono tutti i sottoinsiemi $X \subset 1, ..., N$ dove N è il numero dei player tale che X è compatibile, ovvero il sistema non sta consegnando lo stesso oggetto a più player. F rappresenta tutte le possibili soluzioni, ma il sistema vuole scegliere la soluzione che massimizza la soddisfazione dei vincitori.
Per quanto riguarda il player invece, lo scopo è:

- Se il player i vince (il suo sottoinsieme o un sovrainsieme), e deve pagare p, la sua soddisfazione è pari a $u_i = t_i - p$
- Se il player i non vince, la sua soddisfazione è pari a $u_i = 0$

Formalmente

- Input del problema:
  - n player egoisti, m oggetti indivisibili
  - Ogni player i vuole un sottoinsieme S_i degli oggetti (questo dato è pubblico)
  - Tipo del player privato i: t_i, rappresenta il massimo valore in denaro che il player è disposto a giocare per vincere il sottoinsieme S_i.
  - Valutazione del player i di $X \in F$: $v_i(t_i, X) = t_i$ se $i \in X$, 0 altrimenti.
- Soluzione: $X \subset 1...N$ tale che X è compatibile ovvero $\forall i, j \in X$ con $x \neq j$, si ha che $S_i \cap S_j = \emptyset$
- Obiettivo: realizzare un meccanismo truthful che massimizzi la somma dei tipi t_i: $\sum_{i \in X} t_i$
  - Il meccanismo chiede ai player di riportare i rispettivi v_i
  - Calcola una soluzione usando l'algoritmo $g()$
  - Tramite un apposito schema di pagamento, chiede al player i vincitore il denaro da consegnare per ottenere il sottoinsieme S_i vinto.

4.10 Progettazione del meccanismo truthful

Affinché un problema si possa definire utilitario, il sistema deve calcolare una soluzione, tra tutte quelle ammissibili, che massimizza la somma delle valutazioni. Presa una soluzione X, il valore totale di X è pari a $\sum_{i \in X} t_i$. Calcolando:

$$\sum_{i \in X} v_i(t_i, X) \tag{4.41}$$

Si ottiene proprio il valore totale di X, pertanto il problema è utilitario e quindi è possibile applicare i Meccanismi VCG.

4.10.1 Applicazione del meccanismo VGC

Il meccanismo VCG consiste di un algoritmo e di uno schema di pagamento $M =< g(r), p(x) >$ così definiti:

- $g(r)$:

$$x* = argmax_{x \in F} \sum_j v_j(r_j, x) \tag{4.42}$$

- $p_i(x*)$: per ogni player i:

$$p_i(x*) = \sum_{j \neq i} v_j(r_j, g(r_{-i})) - \sum_{j \neq i} v_j(r_j, x*) \tag{4.43}$$

$g(r)$ è un algoritmo che deve calcolare, tra tutte le soluzioni ammissibili, la soluzione ottima ovvero quella che dà il valore totale massimo. Il problema è dovuto al fatto che **non è possibile calcolare in tempo polinomiale la soluzione ottima** poiché l'algoritmo è NP-Hard (a meno che P = NP). Inoltre il Meccanismo VCG funziona esclusivamente se si utilizza quell'algoritmo con quello schema di pagamento: sostituendo l'algoritmo con un algoritmo approssimato o con una euristica, e si assegnano i pagamenti in modo analogo a quanto avviene nel meccanismo VCG, il nuovo meccanismo **non è veritiero**.
Vale in realtà un risultato ancora più forte: in tempo polinomiale non è possibile calcolare un'approssimazione dell'ottimo che sia sotto un fattore dell'ordine di (circa) $\sqrt{m}$, dove m rappresenta il numero degli oggetti.

Teorema 16. *Il calcolo dell'approssimazione dell'asta combinatorica che sia migliore di un fattore $m^{\frac{1}{2}-\epsilon}$ è un problema NP-Hard, per ogni $\epsilon > 0$*

Dimostrazione. **Richiami**

Un algoritmo di approssimazione per un problema di ottimizzazione è un algoritmo polinomiale che, nel caso peggiore, trova sempre una soluzione non ottima approssimata. Se quindi l'algoritmo è ρ-approssimato allora la soluzione calcolata dall'algoritmo si discosta in senso moltiplicativo al più ρ dall'ottimo. Ovvero $\frac{OPT}{soluzione} \leq \rho$. Tanto più ρ è vicino a 1, tanto più l'algoritmo di approssimazione è buono.

Nel caso del problema dell'asta combinatorica, il risultato di hardness che verrà dimostrato a breve afferma che un algoritmo che restituisce una soluzione

che garantisce sempre che $\frac{OPT}{soluzione} \leq \sqrt{m}$ è un problema NP-Hard.
Per dimostrare il teorema, si effettua una riduzione dal problema del Max Indipendent Set (massimo insieme indipendente).

Max Indipendent Set

Sia $G(V, E)$ un grafo non orientato e non pesato, si vuole trovare un insieme indipendente di cardinalità massima. Un insieme indipendente è un insieme di nodi $U \subset V$ tale che non esistono archi tra questi nodi. Ne consegue che due nodi sono indipendenti tra loro se non sono collegati da un arco. **Esempio:** Sia $v \in V$ un nodo qualsiasi del grafo G, v rappresenta un insieme indipendente. La soluzione appena riportata quanto è buona rispetto ad un potenziale ottimo? Nel caso peggiore, l'ottimo può essere al più n, ovvero tutti i nodi. Si può però dimostrare il seguente risultato:

Teorema 17. *Il problema del Max Indipendent Set non si può approssimare meglio di $n^{1-\epsilon}$, per ogni $\epsilon > 0$.*

Un algoritmo che sceglie un solo nodo del grafo è una n-approssimazione, se si vuole fare $n^{1-\epsilon}$, con ϵ anche piccolissimo, non si può fare.

Riduzione
Sia $G(V, E)$ l'istanza di Max Indipendent Set, si costruisce l'istanza dell'asta combinatorica in questo modo:

- I player sono i nodi, gli archi sono gli oggetti.
- Obiettivo: ogni player vuole tutti gli archi a lui incidenti.
- La riduzione è valida anche quando $t_i = 1$

Se, quindi, si vuole che i vincitori del gioco siano due, significa che ai due vincitori non è possibile dare lo stesso oggetto, e quindi sono un insieme indipendente. Se, infatti, si fa vincere un determinato player/nodo, al vincitore spettano tutti gli archi incidenti.
E' quindi possibile affermare che l'istanza dell'asta combinatorica ha una soluzione di valore $\geq k$ se e soltanto se esiste un insieme indipendente nel grafo di dimensione $\geq k$.
Per dimostrare il risultato di hardness, si assuma dunque che si ha un algoritmo che in tempo polinomiale calcola una soluzione di valore k per l'istanza dell'asta combinatorica, avente la seguente proprietà: è una soluzione approssimata meglio di $m^{\frac{1}{2}-\epsilon}$:

$$\frac{OPT_{CA}}{k} \leq m^{\frac{1}{2}-\epsilon} \tag{4.44}$$

Se ciò fosse vero, l'algoritmo si può utilizzare per calcolare dei buoni insiemi indipendenti di un grafo. Questo implicherebbe quindi anche una soluzione di valore k per l'istanza dell'Indipendent Set tale che:

$$\frac{OPT_{IS}}{k} = \frac{OPT_{CA}}{k} \leq m^{\frac{1}{2}-\epsilon} \leq n^{1-2\epsilon} \tag{4.45}$$

Poiché la corrispondenza è 1:1 tra asta combinatorica e insieme indipendente, le soluzioni dei due problemi coincidono. Si tenga in considerazione il fatto che in un grafo G non orientato il numero degli archi può essere $m \leq n^2$ e, quindi, si può trovare una soluzione approssimata meglio di $n^{1-2\epsilon}$. Questo è in **contrasto** con il risultato di hardness riportato precedentemente: il problema del Max Indipendent Set non si può approssimare meglio di $n^{1-\epsilon}$, per ogni $\epsilon > 0$.

□

4.10.2 Applicazione del meccanismo One Parameter

Il problema dell'asta combinatorica è utilitario e, come già visto, il meccanismo VCG non può essere implementato in tempo polinomiale. Esiste però un'altra classe di meccanismi che possiamo implementare su problemi utilitari: i meccanismi One Parameter.
Le condizioni affinché un problema si possa definire One Parameter Binary Demand sono le seguenti:

- Il tipo privato di un player a_i è un singolo parametro $t_i \in R$.
- La funzione di valutazione del player a_i ha la forma: $v_i(t_i, o) = t_i w_i(o)$ dove $w_i(o) \in 0, 1$ rappresenta il carico di lavoro per in player a_i in o
- Quando $w_i(o) = 1$, allora a_i è selezionato in o

Poiché nell'asta combinatorica la valutazione di un player a_i è della forma $v_i(t_i, o) = t_i$ se il player a_i vince oppure $v_i(t_i, o) = 0$ se il player a_i non vince, è possibile descrivere la valutazione di un player a_i con $v_i(t_i, o) = t_i * w_i(o)$ dove $w_i(o) = 0 \vee 1$ come nei problemi Binary Demand appena descritti. Si ponga **attenzione** al fatto che il carico di lavoro è una conseguenza dell'algoritmo che viene scelto, quindi non è possibile pianificare un carico di lavoro monotono.

Progettare un meccanismo veritiero per un problema One Parameter, significa progettare un algoritmo monotono per il problema di ottimizzazione soggiacente, e accompagnare tale algoritmo con uno schema di pagamento opportuno. Quindi per trovare un buon meccanismo veritiero bisogna trovare un buon algoritmo monotono. Un algoritmo si definisce **monotono** per i problemi Binary Demand di massimizzazione se:
$\forall$ player a_i, per ogni possibile dichiarazione $r_{-i} = (r_1, ..., r_{i-1}, r_{i+1}, ..., r_N)$, il carico di lavoro $w_i(g(r_{-i}, r_i))$ ha la forma seguente:

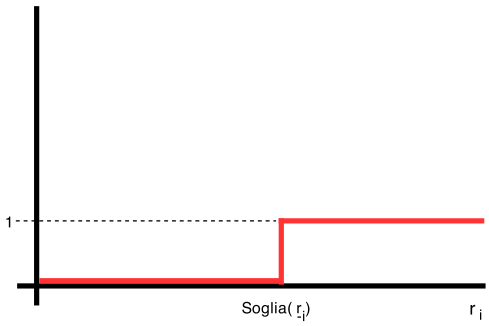


Figura 4.19: Soglia

Ovvero se il player dichiara 0, l'algoritmo assegna carico di lavoro pari a 0 poiché il player non viene selezionato (non vince), se la dichiarazione di un player è pari o superiore invece a una determinata soglia Θ_i, che dipende solo dalle dicharazioni degli altri player, l'algoritmo assegna gli oggetti al player. Formalmente la soglia è dunque:

$$\Theta_i(r_{-i}) \in R \cup +\infty$$

Se la soglia è infinito, il player non vince mai, se invece è pari a 0 vince sempre. Se infine il pagamento per il player a_i è pari $p_i(r) = \Theta_i(r_{-i})$ allora il meccanismo è truthful e l'utilità del player a_i è.

$$u_i = t_i - \Theta_i(r_{-i})$$

Obiettivo: non essendo possibile progettare un meccanismo VCG (è NP-Hard come visto), lo scopo è progettare un algoritmo $g()$ monotono. Se un algoritmo è monotono è possibile dar vita ad un meccanismo veritiero, e più l'algoritmo è buono e monotono, più si riesce a progettare un meccanismo veritiero e buono. Quindi schematizzando:

- $g()$ è monotono
- La soluzione tornata da $g()$ è una buona soluzione, ad esempio una soluzione approssimata
- $g()$ e $p()$ sono calcolabili in tempo polinomiale.

Per garantire la terza condizione, si ricordi il risultato di hardness: non è possibile fare meglio di $m^{\frac{1}{2}-\epsilon}$

Algoritmo greedy $\sqrt{m}$-approssimato

Passi dell'algoritmo:

1. Dati n player e m oggetti, riordina le dichiarazioni dei player secondo la funzione:
$$\frac{v_1}{\sqrt{|S_1|}} \geq \frac{v_2}{\sqrt{|S_2|}} \geq ... \geq \frac{v_n}{\sqrt{|S_n|}} \tag{4.46}$$
Ovvero più la valutazione di un player è alta, maggiore sarà la probabilità che finisca nell'ordinamento tra le prime posizioni. Ma più la cardinalità

dell'insieme di oggetti di un player è grande, maggiore sarà la probabilità che finisca nell'ordinamento tra le ultime posizioni. In sostanza, si usa tale funzione per bilanciare questi due principi cioè soddisfare i player con una valutazione alta e i player che hanno un sottoinsieme di oggetti di cardinalità piccola.

2. $W = \emptyset; X = \emptyset$
3. for i = 1 to n do
   (a) if $S_i \cap X = \emptyset$ then $W = W \cup i; X = X \cup S_i$
4. return W

Lemma 8. *L'algoritmo g() è monotono*

Dimostrazione. Si assuma che un player a_i vince per un'istanza del problema. Se il player a_i aumenta v_i, il player continua a vincere? Se la risposta è sì allora l'algoritmo è monotono. Per dimostrare che la risposta è affermativa, prendiamo un player a_i che vince, aumentiamo la valutazione v_i ed eseguiamo nuovamente l'algoritmo: l'algoritmo può decidere di lasciare il player, nell'ordinamento, nella stessa posizione della valutazione precedente o, addirittura, può spostarlo tra le prime posizioni. E, quindi, se vinceva con la valutazione precedente e, quindi, era compatibile, vince necessariamente anche con la nuova valutazione, poiché rimane compatibile. □

Schema di pagamento I pagamenti vanno calcolati solo per i player che hanno vinto: si deve calcolare, in tempo polinomiale, la soglia di uscita ovvero il valore più piccolo sotto il quale, nel caso in cui il player dichiarasse sotto quel valore, il player non vincerebbe.
Si esamini il comportamento dell'algoritmo, si consideri un player a_i che ha vinto e si abbassi la valutazione v_i. Ovvero:

$$\frac{v_1}{\sqrt{|S_1|}} \geq ... \geq \frac{v_i}{\sqrt{|S_i|}} \geq ... \geq \frac{v_n}{\sqrt{|S_n|}}$$

Con la nuova valutazione abbassata:

$$\frac{v_1}{\sqrt{|S_1|}} \geq ... \geq ... \geq \frac{v_i}{\sqrt{|S_i|}} \geq \frac{v_n}{\sqrt{|S_n|}}$$

Quindi ogni volta che si riduce il valore della valutazione, il player a_i scende di una posizione nell'ordinamento, fino ad arrivare alla coda.
Si elimini quindi il player a_i e si riesegua l'algoritmo, l'algoritmo inizierà a selezionare un determinato insieme di player vincitori in ordine. Consideriamo il primo indice j selezionato a destra di i tale che $S_j \cap S_i \neq \emptyset$. In altre parole, il player a_i non è più vincitore non appena si seleziona come nuovo vincitore un player a_j che rende il player a_i non compatibile.
La soglia dipenderà dunque dalla posizione di j: il player i si troverà nella posizione j, ammesso che j esiste, quando:

$$\frac{v_i}{\sqrt{|S_i|}} = \frac{v_j}{\sqrt{|S_j|}}$$

Da cui segue:

$$v_i = v_j * \frac{\sqrt{|S_i|}}{\sqrt{|S_j|}}$$

Quindi:

$$p_i = v_j * \frac{\sqrt{|S_i|}}{\sqrt{|S_j|}}$$

$$p_i = 0 \text{ se j non esiste}$$

Lemma 9. *Sia OPT la soluzione ottima per l'asta combinatorica, e sia W la soluzione calcolata dall'algoritmo $\sqrt{m}$-approssimato, allora nel caso peggiore il valore dell'ottimo (chiaramente è più grande della soluzione calcolata dall'algoritmo) è più grande al più di un fattore $\sqrt{m}$. Dato il risultato di hardness, questo è la migliore approssimazione che si può ottenere.*

$$\sum_{i \in OPT} v_i \leq \sqrt{m} \sum_{i \in W} v_i \tag{4.47}$$

Dimostrazione. Si vuole far vedere che, quando l'algoritmo greedy seleziona un determinato player i, non sta perdendo troppo rispetto a quello che si sarebbe selezionato nell'ottimo.
Sia W la soluzione calcolata, sia OPT la soluzione ottima. Per ogni $i \in W$ definiamo:

$$OPT_i = \{j \in OPT : j \geq i \wedge S_j \cap S_i \neq \emptyset\}$$

Gli j rappresentano i vincitori dell'ottimo che l'algoritmo greedy non ha fatto vincere perchè ha selezionato i come vincitore. In altre parole, si prendono tutti gli indici j dell'ottimo che sono dopo i nell'ordinamento e che l'algoritmo non ha fatto vincere perchè ha scelto i come vincitore. Vale quindi:

$$\cup_{i \in W} OPT_i = OPT$$

E' sufficiente provare che:

$$\sum_{j \in OPT_i} \leq \sqrt{m} v_i; \forall i \in W$$

Supponiamo che ciò sia vero, allora valutiamo la disequazione per tutti i player i ottenendo:

$$\sum_i \sum_{j \in OPT_i} v_j \leq \sqrt{m} \sum_{i \in W} v_i \tag{4.48}$$

Poiché in OPT_i c'è almeno un elemento dell'ottimo, e dal momento che gli stessi elementi possono anche essere presi più volte (un determinato valore dell'ottimo può far parte di più OPT_i) abbiamo trovato una maggiorazione per la soluzione ottima, che è al più $\sqrt{m}$ volte la soluzione approssimata:

$$\sum_{i \in OPT} v_i \leq \sum_i \sum_{j \in OPT_i} v_j \leq \sqrt{m} \sum_{i \in W} v_i \tag{4.49}$$

L'osservazione cruciale da fare è la seguente: poichè ho selezionato il player i prima di tutti gli j di OPT_i allora l'appetibilità di i è almeno O dell'appetibilità degli j in OPT_i. Ovvero $\forall j \in OPT_i$:

$$v_j \leq v_i * \frac{\sqrt{|S_j|}}{\sqrt{|S_i|}}$$

Unendo tale considerazione con quella precedente, otteniamo:

$$\frac{v_i}{\sqrt{|S_i|}} \sum_{j \in OPT_i} \sqrt{|S_j|} \leq \sqrt{m} v_i \tag{4.50}$$

E' necessario dunque stimare $\sum_{j \in OPT_i} \sqrt{|S_j|}$ e per questo si farà uso della disuguaglianza di Cauchy Schwarz:

$$\sum_{j \in OPT_i} \sqrt{|S_j|} \leq \sqrt{|OPT_i|} \sqrt{\sum_{j \in OPT_i} |S_j|} \leq \sqrt{|S_i|}\sqrt{m} \tag{4.51}$$

Si può banalmente osservare che $\sqrt{\sum_{j \in OPT_i} |S_j|} \leq m$ (dove m sono gli oggetti), ma anche $|OPT_i| \leq |S_i|$ (si noti che gli j dell'ottimo sono disgiunti e nel caso peggiore ogni elemento in S_i è posseduto anche da ogni singolo j)

□

4.11 Esercizi

Esercizio 1 Progettare un meccanismo veritiero per il porblema del calcolo del cammino minimo tra due nodi nello scenario in cui un agente controlla più di un arco del grafo. Come nel caso del singolo arco, i pesi degli archi controllati dall'agente i sono privati e la valutazione di un agente rispetto a una soluzione (cammino) P è uguale alla somma dei veri pesi degli archi selezionati nel cammino.

Esercizio 2 Pensare allo scenario in cui un agente controlla più di un arco del grafo e si vuole progettare un meccanismo veritiero per calcolare un SPT. Come nel caso del singolo arco, i pesi degli archi controllati dall'agente i sono privati e la valutazione di un agente rispetto a una soluzione (albero) T è uguale alla soma dei veri pesi degli archi selezionati nel cammino. E' possibile usare i meccanismi VCG? E' possibile usare i meccanismo One Parameter?

Capitolo 5

Stackelberg Game

5.1 Min Spanning Tree

Dato un grafo $G(V, E)$ non diretto e pesato, per calcolare il Min Spanning Tree T bisogna trovare lo Spanning Tree di G, che è un albero che ricopre tutti i vertici del grafo, che minimizza il peso totale:

$$T(V, F) \text{con } F \subset E$$

$$\sum_{e \in F} w(e)$$

Tra gli algoritmi noti si consideri l'algoritmo di Kruskal (1956). Breve spiegazione:

- Sia T un albero vuoto.
- Si ordinano gli archi di G in ordine crescente di peso.
- Si aggiunge l'arco e a T se e solo se non forma un ciclo con gli archi precedentemente selezionati.

Esempio

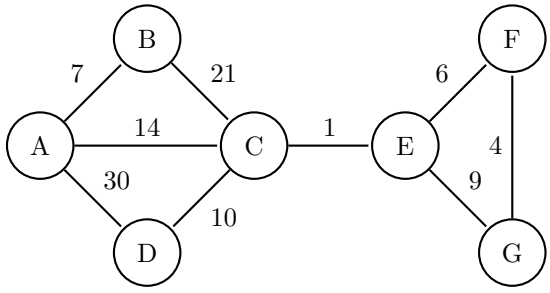


Figura 5.1: G(V, E)

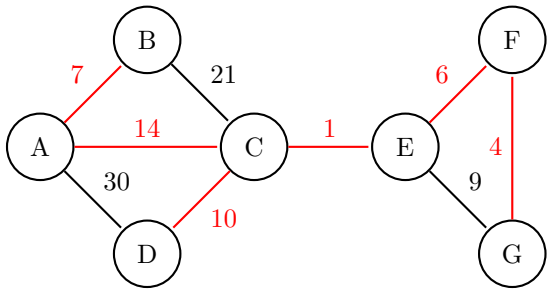


Figura 5.2: T

5.2 Descrizione del gioco

E' un problema puramente algoritmico di ottimizzazione che nasce in uno scenario di Game Theory. Prevede le seguenti proprietà e caratteristiche:

- Partecipano al gioco due player: leader e follower
- Il gioco sequenziale: prima gioca il leader, poi il follower, e via discorrendo.
- Il leader sa cosa farà il follower e ottimizza una data funzione obiettivo che dipende dalla scelta del follower. Il follower, al suo turno, conosce la scelta del leader e ottimizza una data funzione obiettivo.
- Obiettivo: trovare una buona strategia per il leader.

Si tratta di un gioco di prezzatura che avviene su un grafo $G(V, E)$ non orientato dove $E = R \cup B$, ovvero gli archi sono partizionati in due catetorie: Rossi e Blu.

- Gli archi $e \in R$ hanno un costo fisso $c(e)$
- gli archi $e \in B$ sono gli archi che possiede il leader.
- Il leader ha il compito di prezzare $(p(e))$ gli archi $e \in B$.
- Prezzati gli archi blu con $p(e)$, il follower osserva il grafo pesato e compra un Min Spanning Tree T di $G(V, E)$
- Comprato T, per ogni arco $e \in B$ selezionato in T il leader guadagna (revenue) la somma dei prezzi stabiliti. Formalmente:

$$\sum_{e \in E(T) \cap B} p(e) \tag{5.1}$$

  Intuitivamente, più il leader alza il prezzo degli archi, maggiore sarà il guadagno ma, conseguentemente, maggiore sarà la probabilità che gli archi del leader non vengano selezionati dal follower.

- Obiettivo del leader: massimizzare la revenue e, quindi, trovare dei buoni algoritmi di prezzatura.

Esempio 1

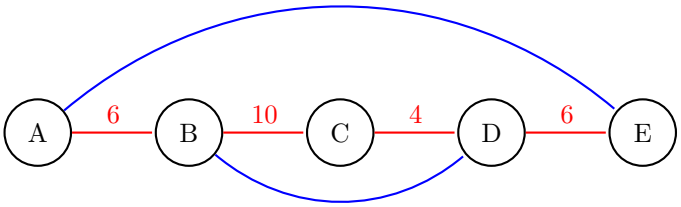


Figura 5.3: G(V, E)

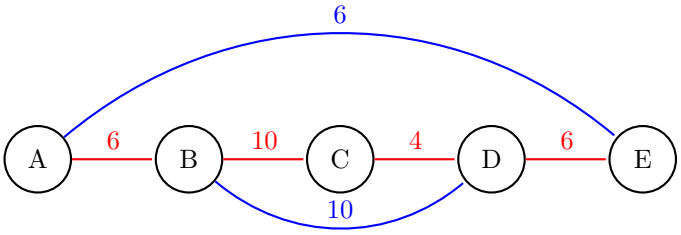


Figura 5.4: Possibile prezzatura

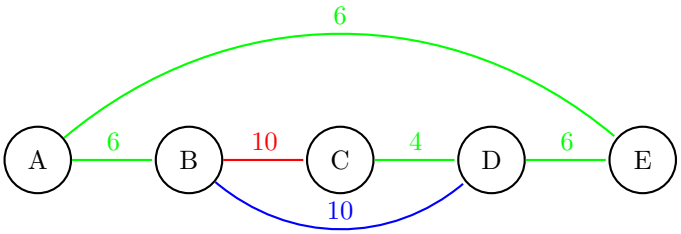


Figura 5.5: T - Revenue = 6

Esempio 2

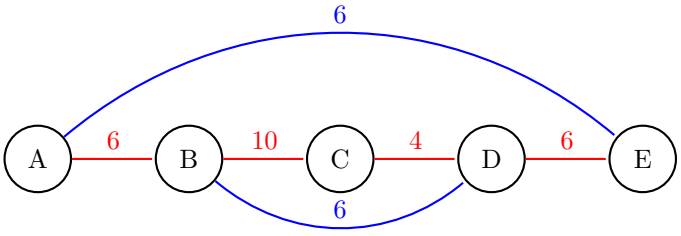


Figura 5.6: Possibile prezzatura

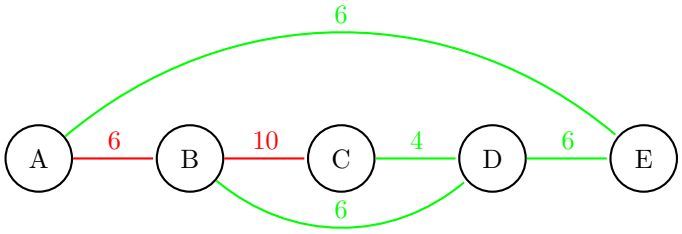


Figura 5.7: T - Revenue = 12

Esempio 4 Sia $G(V, E)$ il grafo in figura **??**

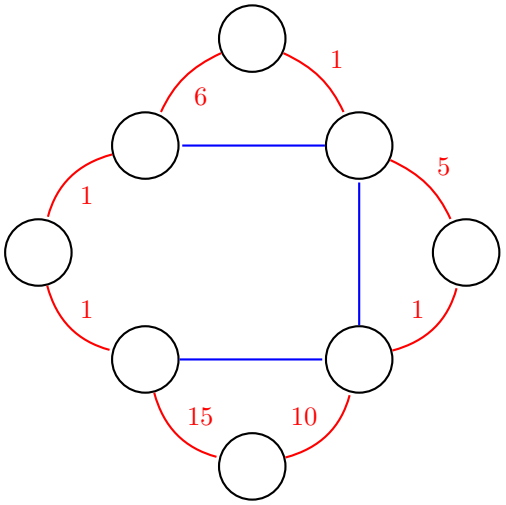


Figura 5.8: G(V, E)

Iniziamo a considerare dapprima gli archi blu singolarmente, con lo scopo di ottenere la maggiore revenue da ogni arco blu, ponendo i prezzi dei restanti a infinito.

Primo arco

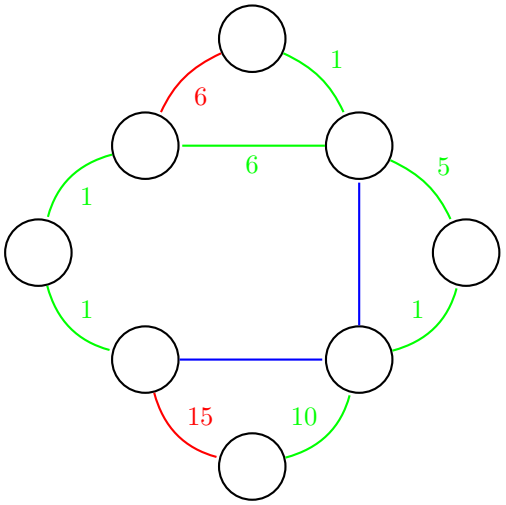


Figura 5.9: G(V, E)

Secondo arco

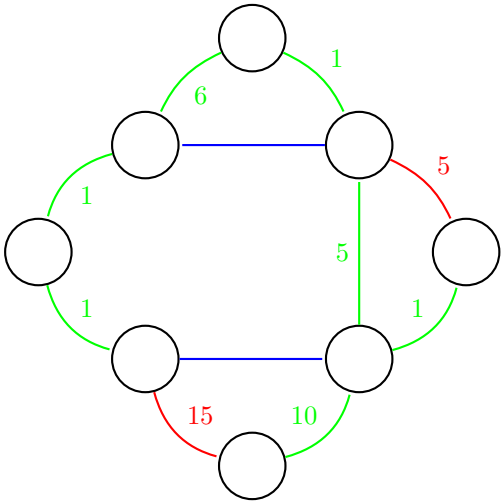


Figura 5.10: G(V, E)

Terzo arco

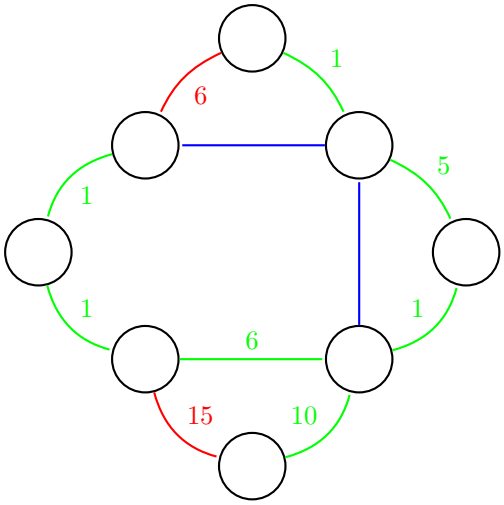


Figura 5.11: G(V, E)

Conclusione

Come si può notare, non è possibile prendere i tre archi blu contemporaneamente perchè uno dei due archi di costo 6 formerebbe un ciclo. Quindi la migliore soluzione è rappresenta in figura **??** con Revenue = 11.

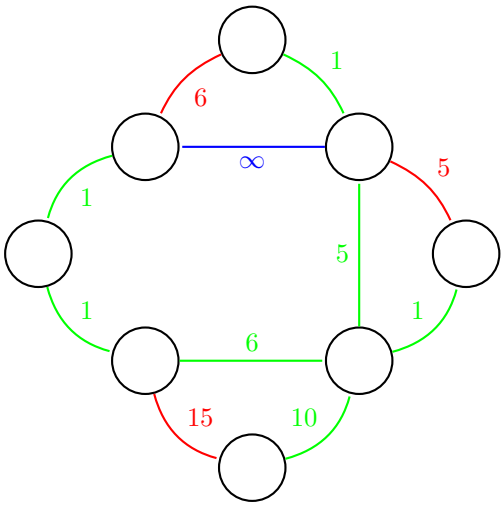


Figura 5.12: T

Assunzioni

- G contiene uno Spanning Tree i cui archi sono tutti rossi. Se G non contenesse un ST di questo tipo, allora ci sarebbe almeno un arco blu, un cut-edge, che sarebbe essenziale (il leader lo prezzerebbe ∞).
- Quando il follower deve scegliere due archi con lo stesso peso, dà sempre la precedenza agli archi blu. Altrimenti, invece di prezzare un arco blu r (soglia oltre la quale l'arco non verrebbe scelto), lo prezzerebbe $r - \epsilon$ e verrebbe scelto dal follower, ottenendo come revenue $r - \epsilon$. Si tratta, semplicemente, di una assunzione di comodo.

Definizione
La revenue del leader dipende esclusivamentre dalla funzione di prezzatura p e non dal particolare MST calcolato dal follower. *Spiegazione*

- Siano $w_1 < w_2 < ... < w_h$ i differenti pesi degli archi in ordine crescente.
- Consideriamo l'algoritmo di Kruskal che, come già osservato, esegue h fasi. In ogni fase i, l'algoritmo considera:
  - Prima tutti gli archi blu di peso w_i (se esistono)
  - Dopo tutti gli archi rossi di peso w_i e li aggiunge se non formano un ciclo (e se ci sono)
- Il numero di archi blu selezionati di peso w_i non dipende dall'ordine in cui gli archi rossi e blu vengono considerati. Osserviamo la fase i-esima: in questa fase vengono selezionati un determinato numero di archi di colore blu di peso w_i, e la revenue dipende dal numero di archi. Il numero di archi blu è, però, indipendente dall'MST selezionato.

  Questo implica il seguente lemma.

Lemma 10. *In ogni funzione di prezzatura ottima, i prezzi assegnati agli archi blu che appaiono in qualche MST appartengono all'insieme* $\{c(e) : e \in R\}$

Spiegazione Sia $G(V, E)$ un grafo che contiene una serie di archi rossi e blu di peso $w = 4$ e una serie di archi rossi e blu di peso $w = 6$. Se alcuni archi blu vengono prezzati $w = 5$, ma non ci sono archi rossi con questo peso, poiché l'algoritmo di Kruskal è a fasi, ad ogni fase verranno considerati prima gli archi blu e poi gli archi rossi. Ne consegue che conviene aumentare il prezzo degli archi con costo $w = 5$ a $w = 6$ poichè, comunque, nonostante gli archi finiscono nella fase degli archi di costo $w = 6$, l'algoritmo per l'assunzione fatta considera sempre prima gli archi blu rispetto agli archi rossi.

Lemma 11. *Sia p una funzione di prezzo ottima e sia T il corrispondente MST. Supponiamo che esiste un arco rosso e in T e un arco blu f non in T tale che e appartiene all'unico ciclo C formato da* $T + f$. *Allora esiste un arco blu* f' *diverso da f tale che* $c(e) < p(f') \leq p(f)$.

Dimostrazione. Sia G(V, E) il grafo in figura **??**

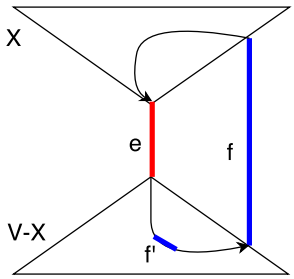


Figura 5.13: G(V, E)

Prima di tutto si osservi che $p(f) > c(e)$ altrimenti l'arco f sarebbe stato selezionato. Si prenda un arco blu $f'(\neq f)$ che ha il peso maggiore nel ciclo C formato da $T + f$. Osserviamo che:

- $p(f') \leq p(f)$ poiché è un arco selezionato.
- $p(f') > c(e)$. Poiché se invece $p(f') \leq c(e)$, si prende f e gli si abbassa il prezzo fino a $c(e)$: in questo nuovo stato, dal momento che f' era l'arco più pesante tra gli archi blu ed era $\leq c(e)$, allora viene preso l'arco f al posto dell'arco e senza perdere alcun altro arco blu già selezionato. Ovvero, il leader sta guadagnando di più e quindi:

  $$p(f) = c(e)$$

  E la prezzatura non poteva essere ottima.

□

5.2.1 Complessità e riduzione

Teorema 18. *Lo Stackelberg game è un problema NP-Hard, anche quando il peso degli archi rossi* $c(e) \in \{1, 2\} \forall e \in R$. *Si effettua la riduzione da Min Set Cover.*

Min Set Cover
Nel problema del Min Set Cover definiamo:

- Input: un insieme di n oggetti $U = \{u_1, ..., u_n\}$; una collezione $S \subset U = \{S_1, ..., S_m\}$
- Output: Tra questi sottoinsiemi si vuole selezionare un insieme di sottoinsiemi C di S tale che copra tutti gli oggetti e il numero di sottoinsiemi selezionato sia di cardinalità minima.

Riduzione
Si parte da un'istanza di Set Cover e, in tempo polinomiale, si genera un'istanza di Stackelberg.

Assunzione: Senza perdita di generalità si può assumere che c'è un oggetto u_n che è presente in tutti gli insiemi. Il motivo per cui assumiamo ciò è dovuto al fatto che se si prende un'istanza difficile di Set Cover e si aggiunge un oggetto fittizio u_n ($u_n \in S_j \forall j$) non si cambia di fatto l'istanza perchè ogni Set Cover nell'istanza precedente è un Set Cover nell'istanza con l'oggetto fittizio e, poichè questo oggetto è presente in tutti gli insiemi e quindi non lo si sta coprendo, ne consegue che ogni Set Cover nell'istanza con l'oggetto fittizio è un Set Cover nell'istanza precedente.

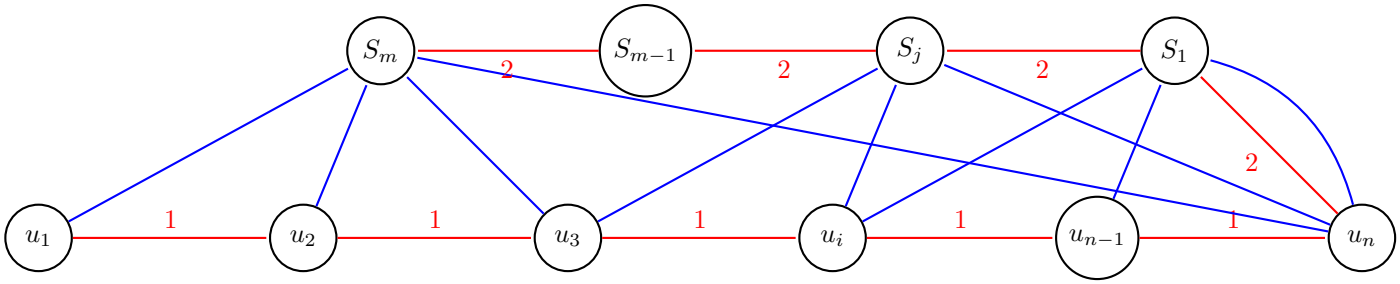


Figura 5.14: G(V, E)

Come si può notare anche dal grafo in figura **??** ogni oggetto in $U = \{u_1, ..., u_n\}$ è rappresentato da un nodo, così ogni sottoinsieme in $S = \{S_1, ..., S_m\}$. Avremo quindi $n+m$ nodi. Inoltre tutti gli oggetti sono collegati da un cammino con archi rossi di peso 1, e tutti i sottoinsiemi sono collegati da un cammino con archi rossi di peso 2. I due insiemi di nodi sono connessi da un arco rosso di peso 2 da u_n a S_1.
Per quanto riguarda gli archi blu invece, esiste un arco blu tra u_i e S_j se e soltanto se $u_i \in S_j$ nell'istanza originale, ovvero solo se S_j copriva u_i.

Teorema 19. *(U, S) ha un cover di dimensione al più t se e soltanto se la massima revenue $r* \geq (n + t - 1) + 2(m - t) = m + 2m - t - 1$.*

Per arrivare alla soluzione, esaminiamo prima i possibili costi degli archi.
Caso 1 Nel caso gli archi blu vengono prezzati 2, l'algoritmo selezionerebbe prima tutti gli archi rossi di costo 1 e, esaminando gli archi di costo 2, per via dell'assunzione precedente, sceglierebbe m archi blu di costo 2 (tanti quanti sono i sottoinsiemi), per una revenue totale di $2m$.

Caso 2 Nel caso gli archi blu vengono prezzati 1, l'algoritmo, per via dell'assunzione precedente, sceglierebbe $n + m - 1$ archi blu (si noti che gli archi blu ricoprono tutti i vertici), per una revenue totale di $n + m - 1$.
Soluzione E' possibile trovare un Set Cover grazie al quale la revenue totale è 1 per ciascun oggetto, 1 per ciascun insieme del Set Cover, 2 per ciascun insieme non del Set Cover. Intuitivamente più il cover sarà piccolo, maggiori saranno gli archi di costo 2 selezionati senza perdere alcun oggetto.

Dimostrazione. $\rightarrow$
Supponiamo di avere un cover di dimensione t, allora la massima revenue è $r* \geq (n + t - 1) + 2(m - t) = m + 2m - t - 1$.
Per ogni arco (u_i, S_j) e S_j è nel cover, allora tutti gli archi incidenti a S_j vengono posti a 1, i restanti vengono posti a 2. Supponiamo che S_1 e S_m siano nel cover, allora (vedi figura **??**):

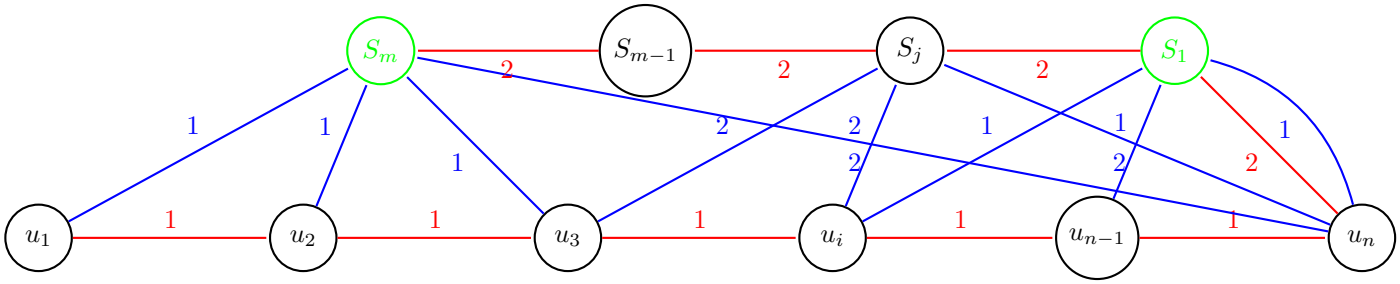


Figura 5.15: G(V, E)

L'algoritmo, una volta eseguito, selezionerà $n + t - 1$ archi blu di costo 1, dove t sono gli insiemi del Set Cover S_m e S_1. Poiché è necessario coprire i restanti insiemi che non sono nel Set Cover, verrà scelto per ciascun insieme un arco blu di costo 2.
Calcolo della revenue: $(n + t - 1) + 2(m - t)$

□

Dimostrazione. $\leftarrow$
Supponiamo di avere un pricing la cui revenue ha un valore $r*$ che dipende dal parametro t, se si ha una soluzione per il problema di Stackelberg che fornisce come risultato la revenue $r*$, allora è possibile riconvertire la soluzione in un cover di size t.
Prima di tutto, tra tutti i possibil pricing ottimi consideriamo i pricing che prezzano gli archi a $p : B = \{1, 2, \infty\}$. Tra tutti i pricing che massimizzano la revenue, consideriamo il pricing tale che l'MST T calcolato dal follower dà il minimo numero di archi rossi. Si farà vedere che c'è sempre un pricing ottimo per cui:

1. T ha solo archi blu
2. E' possibile trovare da tale pricing un Set Cover di dimensione t.

Remark
Supponiamo che in T sono presenti archi rossi solo di peso 1, se si hanno archi (u_i, S_j) blu di peso 2, e uno di questi archi viene selezionato, allora S_j deve essere una foglia in T.

Per dimostrare ciò procediamo per contraddizione. Supponiamo che S_j non sia una foglia, allora necessariamente avrebbe grado > 1. Quindi oltre all'arco blu di costo 2 (u_i, S_j), esisterebbe un altro arco (u_h, S_j) di colore blu (non può essere rosso per via del fatto che nell'istanza tutti gli archi rossi tra oggetto e insieme hanno peso 2, ma abbiamo assunto che nella soluzione T sono presenti solo archi rossi di peso 1).
Tra u_h e u_i esiste però un cammino, composto da soli archi rossi di peso 1: ne consegue che, eseguendo l'algoritmo, verrebbero selezionati tutti gli archi rossi di peso 1 del cammino e infine l'arco (u_i, S_j) blu di peso 2.
Ovvero l'arco (u_h, S_j) non verrebbe mai selezionato e quindi S_j è una foglia.

Dimostrazione 1
Dimostriamo che, se si prende il pricing che minimizza il numero di archi rossi, allora in realtà gli archi rossi non vengono mai selezionati nella soluzione e quindi T ha solo archi blu.
Assumiamo che vi sia un arco rosso e in T (è stato selezionato). Rimuovendo e da T l'albero si divide in due sottoalberi. Osserviamo che dal momento che gli archi blu ricoprono tutto il grafo, e quindi è possibile selezionare almeno un Spanning Tree composto da soli archi blu, allora esiste almeno un arco blu, non selezionato ($f \notin T$), che forma un ciclo C con l'arco e.

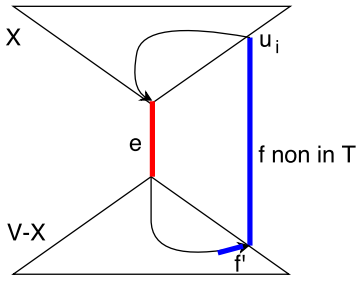


Figura 5.16: G(V, E)

Come già visto nel lemma precedente, esiste un arco blu $f' \in T$ tale che $c(e) < p(f') \leq p(f)$. Questo implica che il prezzo dell'arco e è pari a 1 (non può essere 2, altrimenti il prezzo di f' sarebbe maggiore di 2 - cioè pari a infinito -, e quindi non sarebbe selezionato nella soluzione), e il prezzo di f' è pari a 2.

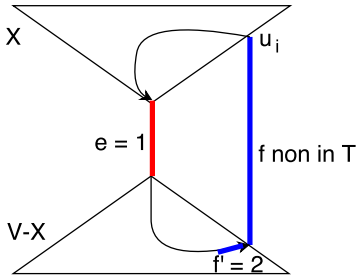


Figura 5.17: G(V, E)

Inoltre l'arco e è il più pesante arco rosso in T, e quindi tutti gli altri archi rossi devono pesare 1. Allora è possibile utilizzare il remark, in cui si affermava

che gli archi di peso 2 selezionati devono essere tali che S_j è una foglia. Attenzione però, nessun arco in $C - \{f, f'\}$ può avere peso 2, altrimenti S_j avrebbe grado almeno 2, e non sarebbe una foglia. Quindi è sufficiente prezzare f e f' 1, cosicché entrambi vengono selezionati e l'arco e escluso dalla soluzione, ottenendo un nuovo MST con la stessa revenue del precedente e meno archi rossi.

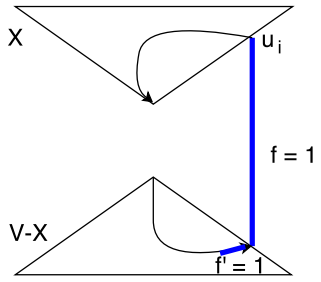


Figura 5.18: G(V, E)

Dimostrazione 2
Dimostriamo che l'albero T composto da soli archi blu induce un Set Cover di dimensione t. Definiamo il Set Cover:

$$C = \{S_j : S_j \text{è collegato a qualche arco blu in } T \text{ con prezzo } 1\}$$

Ogni u_i deve essere incidente in T a qualche arco blu di prezzo 1 e quindi C è un cover.
Ogni $S_j \notin C$ deve essere una foglia in T, e quindi la revenue è:

$$r = n + |C| - 1 + 2(m - |C|) = n + 2m - |C| - 1 \geq n + 2m - t - 1 \quad (5.2)$$

Ovvero:

$$|C| \leq t \quad (5.3)$$

□

5.2.2 Algoritmo single price

Passi dell'algoritmo:

1. Si ordinano in modo crescente gli archi rossi $c_1 < c_2 < ... < c_k$
2. For $i = 1...k$
   - Per ogni $e \in B$, $p(e) = c_i$
   - Controlla la revenue ottenuta
3. Torna la soluzione che restituisce la miglior revenue

Teorema 20. *Sia r la revenue calcolata dall'algoritmo Single Price e sia $r*$ la revenue ottima. Allora*

$$\frac{r*}{r} \leq \rho$$

$$dove\ \rho = 1 + min\{log|B|, log(n-1), log(\frac{c_k}{c_1})\}$$

Dimostrazione. Sia T l'albero calcolato rispetto alla prezzatura ottima.
Siano h_i il numero di archi blu in T con prezzo c_i.
Costruiamo il grafico in figura **??** ponendo sull'asse delle ascisse gli h_i mentre sull'asse delle ordinate gli c_i
Esempio: h_k rappresenta il numero di archi blu con prezzo c_k, e la revenue è data proprio dall'area del rettangolo. La revenue $r*$ è data ovviamente dalla somma delle singole aree.

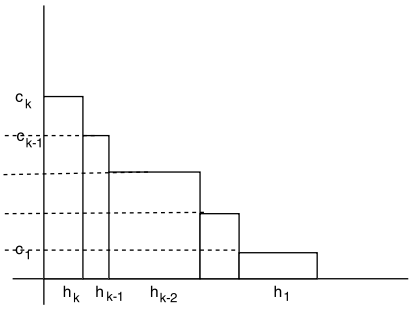


Figura 5.19: Grafico

Ora consideriamo la funzione:

$$f(x) = c * \frac{1}{x}$$

La costante c è scelta in modo tale che la funzione tocca i rettangoli almeno in un punto e tutti i rettangoli si trovano sempre al di sotto della curva.

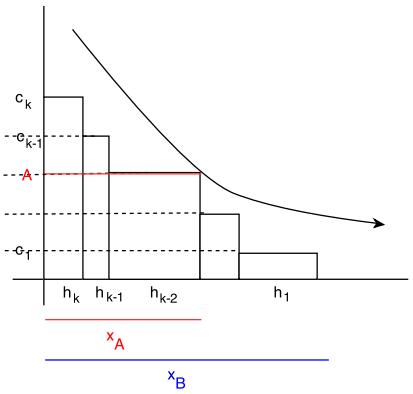


Figura 5.20: Grafico

Per capire cosa è c, si tenga in considerazione la seguente osservazione. Scelto un punto di intersezione, questo coincide con un rettangolo di altezza A, che ha un determinato valore c_j. Quando $x = x_A$, per ottenere la funzione tangente al rettangolo, bisogna avere $f(x_A) = A$.
In sostanza, la costante c deve essere:

$$c = x_A * A$$

E quindi

$$f(x) = x_A * A * \frac{1}{x}$$

In questo modo quando $x = x_A$, $f(x_A) = A$. Si noti che l'area del rettangolo con base x_A e altezza $f(x_A)$ è c.
Osservazione: $c \geq c_k$

L'idea è di dare come **upper bound** della revenue l'integrale da 1 a x_B :

$$r* \leq c + \int_1^{x_B} c * \frac{1}{x} dx = c(1 + logx_B - log1) = c(1 + logx_B)$$

Poiché

$$x_B = \sum_j h_j \leq min\{n-1, |B|\}$$

Per capire il motivo per cui la revenue r del single price è almeno c, ricordiamo brevemente il funzionamento dell'algoritmo. Il single price pone tutti gli archi blu a c_1, poi a c_2 e via discorrendo, fino ad un valore $A = c_j$. Il numero di archi selezionati nell'MST in cui tutti gli archi blu sono posti a $c_j = A$ è pari alla somma di h_k, h_{k-1}, h_{k-2} e quindi la revenue è $x_A * A = c$. E dunque in definitiva:

$$\frac{r*}{r} \leq \frac{c(1 + logx_B)}{c} = 1 + logx_B \tag{5.4}$$

Ora dimostriamo l'ultima parte del teorema, ovvero che il rapporto è $\leq 1 + log(c_k/c_1)$. Per dimostrare ciò è sufficiente invertire il ruolo di x e y nel grafico. Troviamo un nuovo upper bound per $r*$:

$$r* \leq c + \int_{c_1}^{c_k} c * \frac{1}{y} dy = c(1 + logc_k - logc_1) = c(1 + log\frac{c_k}{c_1})$$

Poiché la revenue r del single price è almeno c, allora:

$$\frac{r*}{r} \leq 1 + log\frac{c_k}{c_1} \tag{5.5}$$

□

5.2.3 Esercizi

Esercizio 1 Sia r la revenue del single price, sia $r*$ la revenue ottima. Allora $r * /r \leq k$, dove k è il numero di archi rossi distinti.

Esercizio 2 Trovare un algoritmo polinomiale per il seguente problema: dato un sottinsieme $F \subset B$ aciclico, trovare una prezzatura p tale che:

- Il corrispondente MST T di p contiene esattamente gli archi blu in F. $E(T) \cap B = F$
- Bisogna massimizzare la revenue