Algorithmic Game Theory

Daniele Avolio

A.A. 2023/2024

Contents

List of Theorems

1 Introduzione

Definizione esame: Solitamente lo schema delle lezioni sarà

$$LezioneTeoria \implies LezioneLaboratorio \tag{1}$$

La lezione di Lab sarà fatta praticamente spesso in *Python*.

2 Cos'è la Game Theory — Teoria dei giochi

La **teoria dei giochi** è una disciplina che studia il comportamento decisionale multi-persona, usato per fare predizioni su come **agenti razionali multipli** interagiscono o si comportano in situazioni di *cooperazione* o in situazione di *conflitto*.

Alcune definizioni di termini:

- **Conflitto**: le azioni dei giocatori hanno effetto sugli Algorithmic
- **Cooperazione**: I giocatori possono collaborare per raggiungere un obiettivo
- **Comportamento razionale**: I giocatori vogliono massimizzare la loro *utilità attesa — expected utility*
- **Predizione**: Il nostro obiettivo è sapere cosa faranno i giocatori, utilizzando *solution concepts - concetti di soluzione*

2.1 E cosa significa Algorithmic Game Theory?

Possiamo dire che algorithmic game theory è un punto d'incontro tra **game theory** e **algorithm design** che punta a ***progettare algoritmi*** *che permettono degle strategie in specifici ambienti.*

2.2 Coalition Games

La **coalition game theory** è una branca della game theory che studia le interazioni tra gruppi di giocatori, che **collaborano** per *raggiungere un obiettivo comune.*

Nota - Shapley Values: Il concetto di **Shapley Values** è un concetto che permette di *spiegare*, circa, come un algoritmo di **machine learning** ha preso una decisione. Ad esempio, mostra le *feature* che hanno avuto un impatto maggiore nella decisione finale della predizione. In pratica mostra i vari *Join — Coalizioni* di features.

Quali sono le domande più importanti in questa sezione?

- Quale coalizione è più probabile che venga formata?
- In che modo i giocatori devono dividere il premio? *(Payoff)*

2.3 Non-Cooperative Games

In questo tipo di giochi, i giocatori **non hanno coalizioni** o comunque non ne hanno bisogno.

Alcuni giochi che fanno parte di questa categoria:

- Scacchi
- Sasso-Carta-Forbice
- *Il dilemma del prigioniero*

	Giocatore 1	
Giocatore 2	Collabora	Tradisci
Collabora	(-1,-1)	(-5,0)
Tradisci	(0,-5)	(-3,-3)

Figure 1: Esempio di dilemma del prigioniero

In questo gioco, la **strategia** migliore per il singolo è quella di **tradire** l'altro giocatore, in quanto è quella che massimizza la sua utilità, precisamente andrebbe a **perdere 0 punti**, mentre l'altro giocatore ne perderebbe 5.

2.4 Tree Decomposition

Alcuni problemi sui grafi hanno una complessità di **NP-HARD** su dei grafi arbitrari, e hanno bisogno di alcune soluzioni che avranno implementazioni complesse e **programmazione dinamica.**

2.5 Computational Social Choice

Questa sezione parla di computazione di risultati risultanti da **regole di voto** — **voting rules** e quali problemi ci possono essere nel rappresentare le preferenze dei giocatori.

	Verdetto		
	Evidenza1	Evidenza2	Colpevole
Giudice1	1	0	Innocente
Giudice2	0	1	Innocente
Giudice3	1	1	Colpevole

Figure 2: Esempio di votazione

Maggiore è il numero di persone che votano, maggiore è la probabilità che il risultato sia corretto.

2.6 Mechanism Design

E' un tipo di **reverse game theory**. Invece di analizzare come i giocatori si comportano in un gioco, lo scopo del *mechanism design* è quello di **creare un gioco** per portare i giocatori a ***comportarsi in un modo specifico*** che *vogliamo noi.* Un esempio molto semplice è il *maccanismo di asta di Ebay.* Altro esempio è quello dei *carrelli dei supermecati.* Il fatto di dover utilizzare una moneta per utilizzare il carrello **porta la persona** a dover riportare il carrello nello stesso posto, invece di lasciarlo in un luogo qualsiasi del supermercato.

2.7 Fair Division of Indivisible Goods

In questa sezione si parla di come dividere delle risorse in modo **fair** tra i giocatori. Ok?

2.8 Cake Cutting

In questa sezione si parla di come dividere dei **beni continui** in bsae alle *preferenze dei giocatori.*

1. Fairness
2. Proportionality
3. Envy-freeness

3 Giochi di Coalizione e Concetti di Soluzione

Spesso la **teoria dei giochi** fa riferimento a tipologie di giochi in cui gli agenti **non collaborano**, ma **competono** tra loro. In questi casi si parla di **gioco non cooperativo**.

Parliamo di un ambiente in cui degli agenti interagiscono tra loro, e ogni agente ha un suo **obiettivo** da raggiungere.

Definizione 3.1 (Gioco non cooperativo) *Un gioco non cooperativo è un gioco in cui gli agenti non collaborano tra loro.*

- *Un set di agenti* $N = \{1, \dots, n\}$.
- *Ogni agente* $i \in N$ *ha un set di azioni* S
- *Ogni agnete* $i \in N$ *ha una funzione di utilità* $u_i : S_1 \times S_2 \cdots S_n \to \mathcal{R}$

Definizione 3.2 (Gioco cooperativo) *Il contrario di giochi non cooperativi sono, banalmente,* ***i giochi cooperatvi****, in cui gli agenti* ***collaborano*** *tra loro.*

Domanda: In quale caso le coalizioni appaiono nella teoria dei giochi cooperativi?

- Allocazioni di task
- Allocazione di risorse
- Esperienza degli agenti complementare tra loro

Esempio di gioco cooperativo: Immaginiamo di avere 9 agenti. Ora, gli agenti devono sceliere:

- Con chi allearsi
- Come agire
- Come dividere il premio

Immaginiamo di avere $p_1, p_2, p_3, c_1, c_2, c_3, e_1, e_2, e_3$. Immaginiamo queste 3 coalizioni:

- $C_1\{c_1, e_3, p_3\}$
- $C_2\{c_3, e_2, p_2\}$
- $C_3\{c_2, e_1, p_1\}$

Una **struttura di coalizione** è del tipo: $CS =< C_1, C_2, C_3 >$.

Definiamo anche **il vettore azioni** $a =< a_{c_1}, a_{c_2}, a_{c_3} >$.

Allora possiamo avere un'**allocazione di risorse**

$$u(C_3|a_{c_3}) = 30 \implies Allocazione :< p_1 = 12, c_2 = 3, e_1 = 15 > \tag{2}$$

Quindi, diciamo che nei giochi collaborativo:

- I giocatori **formano coalizioni**
- Ogni coalizione ha associato un **worth**
- Alla fine c'è un **total worth** da distribuire

3.0.1 Tassonomia dei giochi cooperativi

Quando parliamo di giochi cooperativi parliamo di giochi in cui i giocatori tra loro collaborano, fanno azioni insiee, e si formano dei vincoli tra loro. Ma dobbiamo differenziare due tipi di **utility games**

Definizione 3.3 (Transferable Utility Games) *La paga viene data al gruppo e si divide tra loro.*

Definizione 3.4 (Non Transferable Utility Games) *L'azione del gruppo fornisce la paga ai singoli giocatori in modo individuale.*

Esempio di Transferable Utility Games:

Hai N bambini, ognuno dei quali ha una certa quantità di denaro: il bambino i-esimo ha b_i dollari.

Sono in vendita tre tipi di vaschette di gelato:

- Tipo 1 costa \$7 e contiene 500g.
- Tipo 2 costa \$9 e contiene 750g.
- Tipo 3 costa \$11 e contiene 1kg.

I bambini hanno una preferenza per il gelato e non si preoccupano del denaro.

Il risultato ottenuto da ciascun gruppo è la quantità massima di gelato che i membri del gruppo possono acquistare unendo il loro denaro. Il gelato può essere condiviso liberamente all'interno del gruppo.

Formalizzazione dei giochi cooperativi:

Un gioco di utilità trasferibile è una coppia (N, v), dove:

- $N = \{1, \ldots, n\}$ è l'insieme dei giocatori (anche chiamato coalizione grandiosa).
- $v : 2^N \to \mathbb{R}$ è la funzione caratteristica.
- Per ogni sottoinsieme di giocatori C, $v(C)$ è l'importo che i membri di C possono guadagnare lavorando insieme.

Facciamo delle assunzioni. Solitamente diciamo che v è **normalizzato**, cioé $v(\emptyset) = 0$. Ci sono altri due casoi però:

- **Non-negativo**: $v(C) \geq 0$ per ogni $C \subseteq N$.
- **Monotono**: $v(C) \leq v(D)$ per ogni C, D t.c $C \subseteq D$.

Tutto questo non è sempre uguale e dipende sempre dallo scenario.
Esempio del gioco dl gelato: Abbiamo tre giocatori:

1. C con 6
2. M con 4
3. P con 4

Ora, abbiamo 3 tipi di gelato con :

- Gelato 1: w = 500 e p = 7
- Gelato 2: w = 750 e p = 9
- Gelato 3: w = 1000 e p = 11

Cosa possiamo dire? Innanzitutto, **nessuno può comprare niente da solo.** Quindi, se vogliamo che qualcuno compri qualcosa, dobbiamo formare una coalizione. Le domande da fare sono: *Quali azioni dobbiamo compiere? In quale modo ci dividiamo il premio?*.

$$\begin{aligned} v(\emptyset) = v(\{C\}) = v(\{M\}) = v(\{P\}) &= 0 \\ v(\{C, M\}) &= 750 \\ v(\{C, P\}) &= 750 \\ v(\{M, P\}) &= 500 \\ v(\{C, M, P\}) &= 1000 \end{aligned}$$

Definizione 3.5 (Outcome) *Un outcome (o risultato) di un gioco di utilità trasferibile $G = (N, v)$ è una coppia (CS, x), in cui:*

- *$CS = (C_1, \ldots, C_k)$ è una struttura di coalizione, cioè una partizione di N.*
- *$\bigcup_i C_i = N$, $C_i \cap C_j = \emptyset$ per $i \neq j$.*
- *$x = (x_1, \ldots, x_n)$ è un vettore di pagamento che distribuisce il valore di ciascuna coalizione in CS.*
- *$\sum_{i \in C} x_i = v(C)$ per ogni C in CS (Efficienza).*

Supponiamo che $v(\{1,2,3\}) = 9$ *e* $v(\{4,5\}) = 4$.
Quindi, $((1,2,3,4,5),(3,3,3,3,1))$ *è un* ***risultato****.*
Invece, $((1,2,3,4,5),(2,3,2,3,3))$ *non è un risultato.*
I ***trasferimenti tra coalizioni*** *non sono consentiti. Un risultato* $(CS, \underline{x})$ *è chiamato* ***imputazione*** *se soddisfa la* ***razionalità individuale****:* $x_i \geq v(\{i\})$ *per tutti* $i \in N$.

Definizione 3.6 (Giochi superaddittivi) *Un gioco di utilità trasferibile* $G = (N, v)$ *è chiamato superadditivo se* $v(C \cup D) \geq v(C) + v(D)$ *per qualsiasi due coalizioni disgiunte* C *e* D.

Esempio: $v(C) = |C|^2$; $v(C \cup D) = (|C| + |D|)^2 \geq |C|^2 + |D|^2 = v(C) + v(D)$.

Nei ***giochi superadditivi****, due coalizioni possono sempre fondersi senza perdere denaro; quindi, possiamo assumere che i giocatori formino la coalizione grandiosa.*

Praticamente, quando due coalizioni collaborando ottengono un risultato che è almeno pari alla somma dei risultati che avrebbero ottenuto se avessero agito da sole.

Un esempio è il seguente:

$$\begin{aligned} v(\emptyset) = v(\{C\}) = v(\{M\}) = v(\{P\}) &= 0 \\ v(\{C, M\}) &= 750 \\ v(\{C, P\}) &= 750 \\ v(\{M, P\}) &= 500 \\ v(\{C, M, P\}) &= 1000 \end{aligned}$$

In questo caso si vede che è un gioco superadditivo perché la collaborazione porta ad un risultato migliore.

3.1 Concetti di soluzione

Assumiamo che la grande coalizione **N** sia formata. Quindi:

$$x = (x_1, x_2, \ldots, c_n)$$

$$x_i \geq 0 \forall i \in N$$

$$x_1 + x_2 + \ldots x_n = v(N)$$

Considera il gioco del gelato con la seguente funzione caratteristica. Si tratta di un gioco superadditivo in cui gli esiti sono vettori di pagamento (modi per dividere 1000).

Come dovrebbero i giocatori condividere il gelato?

Se lo dividono come (200, 200, 600), Charlie e Marcie possono ottenere più gelato acquistando una vaschetta da 750g da soli e dividendo equamente. L'esito (200, 200, 600) **non è stabile**!

Definizione 3.7 (Core O Nucleo) *Il nucleo o core di un gioco è l'insieme di tutti gli* ***esiti che sono stabili****, cioé quegli esiti che no vengono scartati da nessuna coalizione.*

$$core(G) = \{(CS, \underline{x}) | \sum_{i \in C} x_i \geq v(C) \ for \ any \ C \subseteq N\}$$

Nota: **x(c)** si identifica come la somma dei valori $\sum_{i \in C} x_i$.
Possiamo accorciare in:

- $x \in R^n$ è un nucleo se $x(c) \geq v(c) \forall C \subseteq N$
- x(N) = v(N)

Torniamo al nostro esempio. Ora vediamo il concetto di **nucleo** applicato

- (200, 200, 600) **non è** nel nucleo:
- $v(\{C, M\}) > x_C + x_M$
- (500, 250, 250) **è nel** nucleo:

nessun sottogruppo di giocatori può deviare in modo che ciascun membro del sottogruppo ottenga di più. Un vettore (x_C, x_M, x_P) è nel nucleo se e solo se soddisfa le seguenti condizioni:

- $x_C + x_M \geq v(\{C, M\})$ (stabilità)
- $x_C + x_P \geq v(\{C, P\})$ (stabilità)
- $x_P + x_M \geq v(\{P, M\})$ (stabilità)
- $x_C \geq v(\{C\})$ (razionalità individuale)
- $x_P \geq v(\{P\})$ (razionalità individuale)
- $x_M \geq v(\{M\})$ (razionalità individuale)
- $x_C + x_P + x_M = v(\{C, M, P\})$ (efficienza)

Queste **3 proprietà** sono importanti.

Definizione 3.8 (Giochi con nucleo vuoto) *Il concetto di nucleo è molto attraente come concetto di soluzione. Purtroppo, ci sono alcuni giochi che hanno* ***un nucleo vuoto****.*

Consideriamo il gioco $G = (\{1, 2, 3\}, v)$ con la seguente funzione caratteristica v:

$$v(C) = \begin{cases} 1 & se \ |C| > 1 \\ 0 & altrimenti \end{cases}$$

Considera un risultato $(CS, \underline{x})$:
Supponi che $CS = (\{1\}, \{2\}, \{3\})$.

- In questo caso, la coalizione grandiosa può deviare.
- $x_1 + x_2 + x_3 = v(\{1\}) + v(\{2\}) + v(\{3\}) < v(\{1, 2, 3\})$.

Cioè, **non è stabile**. Guarda la disequazione.
Supponi che $CS = (\{1, 2\}, \{3\})$.

- In questo caso, o 1 o 2 ottengono meno di 1, quindi possono deviare con 3.
- $x_1 + x_3 = x_1 + 0 < 1 < v(\{1, 3\})$.

Collaborando devono **dividere** questo **1** che ottengono. Quindi, 1 o 2 prenderò meno dell'altro e l'altro potrebbe deviare con il 3.
Supponi che $CS = (\{1, 2, 3\})$.

- In questo caso, $x_i > 0$ vale per alcuni i, diciamo $i = 3$.
- Quindi, $x(\{1, 2\}) < 1$, ma $v(\{1, 2\}) = 1$.

Quindi in ogni caso *qualcuno potrebbe cercare una coalizione migliore di quella in cui si trova attualmente.* Questo porta ad avere un **nucleo vuoto**.

3.1.1 Cosa fare quando il nucleo è vuoto?

Questa situazione prende il nome di $\epsilon - Core$. In questo casi si vuole approssimare un esito stabile.
Bisogna *rilassare* il concetto di nucleo:

- Nucleo: $(CS, x) : x(C) \geq v(C)$ per ogni $C \subseteq N$
- ϵ-nucleo: $(CS, x) : x(C) \geq v(C) - \epsilon$ per ogni $C \subseteq N$

Solitamente questa nozione è definita solo per giochi superadditivi.
Per esempio, consideriamo il gioco $G = (\{1, 2, 3\}, v)$, con $v(C) = 1$ se $|C| > 1$, $v(C) = 0$ altrimenti:

- Il $\frac{1}{3}$-nucleo non è vuoto: $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}) \in \frac{1}{3}$-nucleo
- Il ϵ-nucleo è vuoto per qualsiasi $\epsilon < \frac{1}{3}$:
- – $x_i \geq \frac{1}{3}$ per qualche $i = 1, 2, 3$; quindi $x(N\{i\}) \leq \frac{2}{3}$, ma $v(N\{i\}) = 1$.

Definizione 3.9 (Nucleo minimo) *Definiamo $\epsilon^*(G)$ come $\inf\{\epsilon | \epsilon$-core di G non è vuoto$\}$.*

- *Si può dimostrare che $\epsilon^*(G)$-core non è vuoto.*

La definizione di $\epsilon^*(G)$-core è il nucleo minimo di G.

- $\epsilon^*(G)$ *è chiamato il valore del nucleo minimo.*

Nel contesto del gioco $G = (\{1, 2, 3\}, v)$ con la funzione caratteristica $v(C)$ definita come segue:

- $v(C) = 1$ se $|C| > 1$
- $v(C) = 0$ altrimenti
- Il 1/3-core non è vuoto: $(1/3, 1/3, 1/3) \in 1/3$-core
- Il ϵ-core è vuoto per qualsiasi $\epsilon < 1/3$:
  - $x_i \geq 1/3$ per qualche $i = 1, 2, 3$, quindi $x(N\{i\}) \leq 2/3$, ma $v(N\{i\}) = 1$.

3.2 Concetti di soluzione avanzati

Ci sono in particolare due che sono molto importanti: **shapy value** e il **Nucleolus**

Più sofisticate considerazioni sulla stabilità

- **Nucleolus**: Il nucleolus è un concetto utilizzato nella teoria dei giochi cooperativi per valutare la giustizia nella distribuzione dei guadagni tra i giocatori.
- **Bargaining set**: L'insieme di contrattazione è un concetto che si riferisce agli insiemi di risultati in cui i giocatori trovano equo e ragionevole partecipare, dati i poteri di contrattazione.
- **Kernel**: Il nucleo è un sottoinsieme del nucleo in cui i giocatori non possono migliorare il proprio risultato cooperando in modo diverso.

Concetto di fairness

- **Shapley value**: Il valore di Shapley è una soluzione per assegnare un valore a ciascun giocatore in modo equo, tenendo conto del loro contributo marginale a ogni possibile coalizione.
- **Banzhaf index**: L'indice di Banzhaf è una misura del potere di voto di ciascun giocatore in un gioco di voto ponderato.

3.2.1 Nucleolus

Definiamo ora il concetto di *eccesso*

Definizione 3.10 (Eccesso) *E' una misura che indica quanto la coalizione* ***è insoddisfatta.***

$$e(S, x) = v(S) - x(S) \tag{3}$$

Facciamo un esempio numerico:

- v({1}) = v({2}) = v({3}) = 0
- v({1,2}) = v({1,3}) = v({2,3}) = 1
- v({1,2,3}) = 3

Minimizzare la insoddisfazione di ogni possibile coalizione. Applichiamo

- x = (0,0,3) $\implies e(\{1,2\}, x) = v(\{1,2\}) - (x_1 + x_2) = 1 - 0 = 1$
- x = (1,2,0) $\implies e(\{1,2\}, x) = v(\{1,2\}) - (x_1 + x_2) = 1 - 3 = -2$

Ritorniamo alla definizione di Nucleolus.
Definizione da Schmeidler: Il nucleolus $\mathcal{N}(\mathcal{G})$ di un gioco $\mathcal{G}$ è l'insieme:

$$\mathcal{N}(\mathcal{G}) = \{x \in \mathcal{X}(\mathcal{G}) | \nexists y \in \mathcal{X}(\mathcal{G}) \; t.c \; \theta(y) \prec \theta(X)\}$$

Che significa proprio che non esiste un altro vettore di pagamento che è preferito da tutti i giocatori rispetto a x. Quindi, il nucleolus è un concetto di soluzione che è **stabile**.

3.3 Shapley Value

Stability vs Fairness
Consideriamo il gioco $G = (\{1,2\}, v)$ con le seguenti caratteristiche:

- $v(\emptyset) = 0$
- $v(\{1\}) = v(\{2\}) = 5$
- $v(\{1,2\}) = 20$

Nel nucleo del gioco, abbiamo l'allocazione $(15, 5)$. In altre parole, il giocatore 1 riceve 15 e il giocatore 2 riceve 5. È importante notare che il nucleo rappresenta una situazione in cui nessun giocatore può ottenere un risultato migliore deviando unilateralmente. In questo caso, il giocatore 2 non può ottenere un risultato migliore deviando.

La domanda principale è: l'allocazione (15, 5) è equa? La giustizia in un contesto di gioco può essere soggettiva e dipendere dalle aspettative e dagli accordi tra i giocatori. Quindi, se l'allocazione è considerata equa o meno potrebbe variare in base al contesto e alle aspettative dei giocatori.

No! Poiché 1 e 2 sono risultati simmetrici nel core possono essere ingiusti! Come facciamo a dividere i pagamenti in modo equo?

Pensiamo a questo. Un risultato equo dovrebbe premiare ciascun agente in base al loro contributo. Nel primo tentativo, dato un gioco $G = (N, v)$, impostiamo $x_i = v(\{1, \ldots, i-1, i\}) - v(\{1, \ldots, i-1\})$. In altre parole, il pagamento per ciascun giocatore è il loro contributo marginale alla coalizione dei loro predecessori. Otteniamo $x_1 + \ldots + x_n = v(N)$; x è un vettore di pagamento.

Tuttavia, questo metodo non funziona poiché il pagamento di ciascun giocatore dipende dall'ordine. Ad esempio, consideriamo il gioco $G = (\{1, 2\}, v)$, con $v(\emptyset) = 0$, $v(\{1\}) = v(\{2\}) = 5$, e $v(\{1, 2\}) = 20$. In questo caso, $x_1 = v(1) - v(\emptyset) = 5$ e $x_2 = v(\{1, 2\}) - v(\{1\}) = 15$.

Notiamo che i risultati non sono gli stessi, indipendentemente dall'ordine. Pertanto, questa formulazione non produce risultati equi.

Un'idea per eliminare la dipendenza dall'ordine è quella di calcolare una media su tutte le possibili permutazioni degli ordini di arrivo.

Ad esempio, consideriamo il gioco $G = (\{1, 2\}, v)$, con $v(\emptyset) = 0$, $v(\{1\}) = v(\{2\}) = 5$, e $v(\{1, 2\}) = 20$. Iniziamo calcolando i pagamenti per due ordini diversi:

- Per l'ordine (1, 2): $x_1 = v(1) - v(\emptyset) = 5$ e $x_2 = v(\{1, 2\}) - v(\{1\}) = 15$.
- Per l'ordine (2, 1): $y_2 = v(2) - v(\emptyset) = 5$ e $y_1 = v(\{1, 2\}) - v(\{2\}) = 15$.

Ora, calcoliamo i pagamenti mediando tra i due ordini:

- $z_1 = (x_1 + y_1)/2 = (5 + 15)/2 = 10$
- $z_2 = (x_2 + y_2)/2 = (15 + 5)/2 = 10$

Il risultato ottenuto è equo, poiché ciascun giocatore riceve 10, indipendentemente dall'ordine in cui sono considerati.

Una permutazione di $\{1, \ldots, n\}$ è una corrispondenza uno a uno da $\{1, \ldots, n\}$ a se stessa.

Denotiamo con $P(N)$ l'insieme di tutte le permutazioni di N.

Denotiamo con $S_\pi(i)$ l'insieme dei predecessori di i in una permutazione $\pi \in P(N)$.

Per $C \subseteq N$, definiamo $\delta_i(C) = v(C \cup \{i\}) - v(C)$ come il contributo marginale del giocatore i a C.

Il valore di Shapley del giocatore i in un gioco $G = (N, v)$ con $|N| = n$ è dato da:

$$\varphi_i(G) = \frac{1}{n!} \sum_{\pi : \pi \in P(N)} \delta_i(S_\pi(i))$$

Supponiamo di scegliere una permutazione dei giocatori in modo uniforme e casuale tra tutte le possibili permutazioni di N. In questo contesto, il valore di Shapley φ_i rappresenta il contributo marginale atteso del giocatore i alla coalizione dei suoi predecessori.

3.3.1 Proprietà del valore di Shapley

Definizione 3.11 (Proprietà 1) *In qualsiasi gioco G, la somma dei valori di Shapley di tutti i giocatori, ossia $\varphi_1 + \ldots + \varphi_n$, è uguale al valore totale del gioco $v(N)$.*

Definizione 3.12 (Proprietà 2) *Un giocatore i è definito un giocatore fittizio (dummy) se $v(C) = v(C \cup \{i\})$ per qualsiasi insieme $C \subseteq N$.*

Proposizione: *Se un giocatore i è un giocatore fittizio, allora il suo valore di Shapley φ_i è uguale a 0.*

Definizione 3.13 (Proprietà 3) *Due giocatori i e j sono definiti simmetrici se $v(C \cup \{i\}) = v(C \cup \{j\})$ per qualsiasi insieme $C \subseteq N \setminus \{i, j\}$.*

Proposizione: *Se i e j sono giocatori simmetrici, allora i loro valori di Shapley φ_i e φ_j sono uguali.*

Definizione 3.14 (Proprietà 4) *Siano $G1 = (N, u)$ e $G2 = (N, v)$ due giochi con lo stesso insieme di giocatori. Allora $G = G1 + G2$ è il gioco con l'insieme di giocatori N e la funzione caratteristica w definita come $w(C) = u(C) + v(C)$ per tutti gli insiemi $C \subseteq N$.*

Proposizione: *Il valore di Shapley di un giocatore i nel gioco $G1 + G2$ è uguale alla somma dei valori di Shapley di i nei giochi $G1$ e $G2$, ossia $\varphi_i(G1 + G2) = \varphi_i(G1) + \varphi_i(G2)$.*

Proprietà riassunte:

1. **Efficienza:** $\varphi_1 + \ldots + \varphi_n = v(N)$
2. **Dummy:** Se i è un giocatore fittizio, allora $\varphi_i = 0$
3. **Simmetria:** Se i e j sono giocatori simmetrici, allora $\varphi_i = \varphi_j$
4. **Additività:** $\varphi_i(G1 + G2) = \varphi_i(G1) + \varphi_i(G2)$

Teorema: Il valore di Shapley è l'unico schema di distribuzione dei pagamenti che soddisfa le proprietà 1-4.

E' possibile scrivere la formula anche in questo modo:

$$\phi(i, \nu) = \sum_{C \subseteq N} \frac{(|N| - |C|)! \times (|C| - 1)!}{|N|!} (\nu(C) - \nu(C \setminus \{i\}))$$

4 Problemi con giochi cooperativi

4.1 Il problema dell'esponenzialità

Come abbiamo visto, possiamo rappresentare con:

- $G = (N, v)$ un gioco, con N giocatori e v una funzione di valore
- $v : 2^N \to \mathbb{R}$ una funzione di valore

E abbiamo definito il core come:

- x(N) = v(N)
- x(s) $\geq$ v(s) $\forall S \subseteq N$
- x $\in \mathbb{R}^N$

Ma se dovessimo calcolarci la funzione di valore v, dovremmo calcolarci tutte le possibili combinazioni che sono:

- {}
- {1}, {2}, {3} ... {n}
- {1,2}, {1,3}, {1,4} ... {1,n}, ... {n,n}
- ...

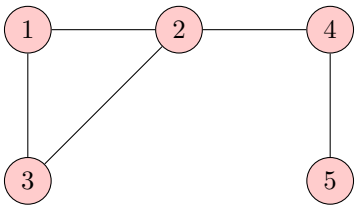
Vediamo che questo valore è **esponenziale**: 2^N. Questo è molto chiaro che funzioni solo da un punto di vista toerico. Nella realtà, gestire problemi di tipo esponenziale non è chiaramente facile e c'è bisogno di alcune soluzioni diverse. Le deiniamo **soluzioni compatte.**

4.2 Le soluzioni compatte

Le **soluzioni compatte** ci permettono di definire il problema in modo più compatto, in modo da poterlo risolvere in modo più efficiente e posssibilmente lineare.

4.2.1 Giochi sui grafi

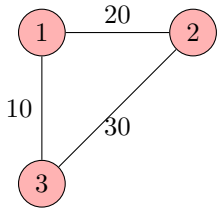
Diciamo innanzitutto che i giochi sui grafi sono **un sotto insieme** dei giochi cooperativi. Immaginiamo di avere questo grafo:



La grandezza di questa struttura è al massimo $|N|^2$.
Il problema è che i giochi a grafo **non sono completi**.
Supponiamo di avere questi valori:

- 1,3 = 10
- 1,2 = 20
- 2,3 = 30
- 1,2,3 = 60

Con un grafo cosi:



Forse ho capito. Se prendiamo un punto nell'insieme dei giochi cooperativi con v({1,2,3}) = -50, allora nel grafo sopra non esiste e quindi non possono rappresentare tutte quante le possibili combinazioni. Quindi non sono completi.

4.2.2 Giochi con contribuzione marginale

I giochi con contribuzione marginale sono un sottoinsieme dei giochi sui grafi.

Ogni gioco sui grafi può essere rappresentato dai giochi a contribuzione marginale.

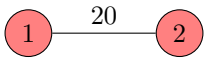
In questo gioco, il valore di una coalizione è dato dalla somma del valore di tutte le regole che si applicano ad una data coalizione.

Se non stai capendo, sotto c'è la lezione di laboratorio che si collega al capitolo 12.6.

Ad esempio:

$$1 \wedge 2 \rightarrow 20$$

E possiamo rappresentarlo come:



E se avessimo:

- $1 \wedge 2 \rightarrow 20$
- $1 \wedge 3 \rightarrow 10$
- $2 \wedge 3 \rightarrow 30$

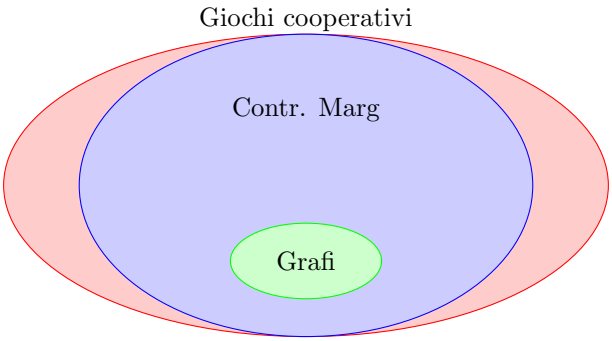
Allora se volessimo rappresentare v({1,2,3}) = -50, dovremmo fare:

- $1 \wedge 2 \wedge 3 \rightarrow -60 - 50$

E allora avremmo:

$$v(\{1,2,3\}) = 20 + 10 + 30 - 60 - 50 = -50$$

E tra l'altro i giochi con contribuzione marginale **corrispondono esattamente** coi giochi cooperativi.



4.3 Giochi di allocazione

Immaginiamo di avere *2 bambini* che hanno *n giocattoli*. Ogni giocattolo ha un valore.

VA RECUPERATA ROBA DALLE SLIDES DOPO NON HO CAPITO

Dato il gioco:

$$G = (N, v), v : 2^N \rightarrow R$$

C'è un problema di allocazione.

Per ogni coalizione C, il valore associato è dato dal valore della migliore coalizione focussandoci sui player solamente in C. Va calcolato il **matching**

massimo in grafico. La cosa importante è che può essere fatto in **tempo polinomiale**.

Cerchiamo di capire cosa vogliamo fare: Siamo in uno **scenario di condivisione**. L'algoritmo considera il migliore modo di costruire le coalizioni per capire lo scenario migliore di come dividere i beni che ci sono in gioco.

4.4 Giochi a voto pesato

Siamo in questa situazione:

- n partiti nel parlamento
- Il partito i ha w_i rappresentanti
- Una coalizione di partiti può formare un governo solamente se la grandezza è almeno q
  - Solitamente $q \geq |\sum_{i=1}^{n} w_i|/2$; maggioranza stretta
- La notazione è $w(C) = \sum_{i \in c} w_i$

Questo può essere descritto come un gioco $G(N, v)$ dove:

- N = {1,2,3, . . . , n}
- v(C) = 1 se $w(C) \geq q$, 0 altrimenti

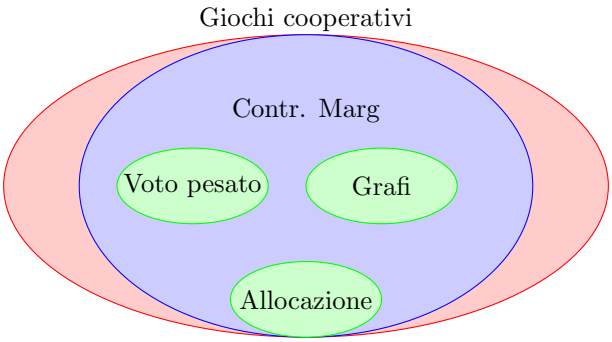
Esempio 4.1

- 1 = 10
- 2 = 20
- 3 = 5
- 4 = 5

Ora, vediamo il valore delle coalizioni:

- v({2,3}) = 1
- v({1,2,3}) = 1
- v({1,2,3,4}) = 1
- v({3,4}) = 0

Anche questi giochi **non sono completi**, ma diciamo che sono **funzionali specificatamente** per alcuni scenari particolari.



Domanda 4.1 *(Puoi calcolare in modo efficiente lo shapley value?)*

La risposta è che **dipende**. In alcuni casi, lo shapley value può essere rappresentato in modo efficiente.

Ma se in alcuni casi la risposta è **NO**, allora dobbiamo **provare** che queste soluzioni sono **intrattabili.**

5 Classi di compplessità dei problemi

Definiamo **problemi facili** i problemi che sono nella classe di complessità **polinomiale: P**

$$\Sigma_o^P = P = \prod_o^P$$

5.1 Problemi in NP

NP è la classe di complessità **non deterministic polinomial: NP**. Il primo problema è:

Definizione 5.1 *(Il problema dei team) Vogliamo fare 2 squadre. L'obiettivo è costruire dei team* ***bilanciati***

Esempio 5.1

- *Giocatore 1 = 3*
- *Giocatore 2 = 9*
- *Giocatore 3 = 1*
- *Giocatore 4 = 7*

Il team bilanciato è:

- *Team 1 = 1 + 9 = 10*
- *Team 2 = 3 + 7 = 10*

Una prima intuizione: Vogliamo controllare una soluzione in un tempo ragionevole. Una macchina di turing non deterministica ha il potere di **guessare** una soluzione possibile.

5.2 Problemi in CO-NP

Con CO-NP classifichiamo i problemi tali che il loro **complemento** sia risolvibile in tempo polinomiale usando una macchina di turing non deterministica. Un problema prototipico: *controllare che una formula booleana non è soddisfacibile.*

5.3 La classe Σ_2^P

Questa è la classe di complessità che contiene i problemi che possono essere risolti in tempo polinomiale da una macchina di turing non deterministica, che usa una macchina di turing non deterministica come un oracolo a costo unitario. Un esempio è *decidere se una* $\forall\exists$ *formula è* ***non soddisfacibile***

5.4 Oltre la gerarchia

Consideriamo $\#P$ la classe di tutte le funzioni che possono essere calcoalte da una **counting turing machine** in tempo polinomiale. Questa **counting turing machine** è una macchina di turing non deterministica che ha un device di output che stampa in binario il numero di computazioni accettabili indotte dall'input.

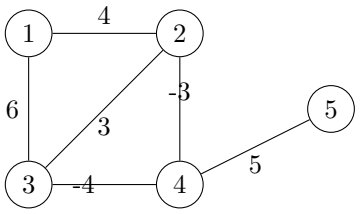
Un esempio è *contrare il numero di assegnamenti di verità che soddisfano una formula booleana.*

PAGINA 121 DA COPIARE

Problema	**Classe di complessità**
CORE	coNP-Complete
BARGAINING SET	$\prod_2^P$-Complete
NUCLEOLUS	Δ_2^P-Complete
SHAPLEY VALUE	$\#P$-Complete

5.5 Core e Hardness

Prendiamo un esempio di grafo:



Ci chiediamo, $C(G) = \emptyset$? Per rispondere a questa domanda, definiamo il core:

- x(n) = v(n)
- $x(S) \geq v(S), S \subseteq N$

Assumiamo che esista un taglio nel grafo, e assumiamo che per contraddizione c'è un punto che è all'interno del core. x(n) = v(n).

Il taglio divide il grafo in S e T.

$$V(S \cup T) = v(S) + v(T) + valoreDelTaglio$$

Nota: Gli agenti vogliono splittare se e solo se *valoreDelTaglio*<0

Quindi, la risposta è; **il core è vuoto se e solo se esiste un taglio negativo**

La dimostrazione, parlando chiaramente, **non l'ho capita lol**.

Controllare che esiste un taglio negativo è un problema **NP-HARD**. Il problema complementare è un **coNP-HARD**

Nota: Massimizzare il taglio è facile, massimo flusso minimo taglio, minimizzare il taglio è difficile.

5.6 Nucleolus e Hardness

Ricordiamo un attimo la definizione del Nucleolus:

Definizione 5.2 *(Nucleolus)*

$$\mathcal{N}(G) = \{x \in X(G) | \nexists y \in X(G) \ t.c \ \theta(y) \preccurlyeq \theta(x)\}$$

Prendiamo una formula $\varphi = (x_1 \vee x_2) \wedge x_3 \vee x_4$. Abbiamo un ordine di indici {1,2,3,. . . ,n}

Assumiamo che le variabili siano *in ordine lessicografico*, quindi in base al loro indice: x_1 è quella meno significativa.

Obiettivo: Trovare l'assegnamento di verità massimo lessicograficamente.

Esempio:

- x_2, x_3 = OK
- x_1, x_4 = NO
- x_1, x_3 = OK

In questo caso, quello che massimizza è x_2, x_3 perché, con pareggio per x_3, abbiamo x_2 che ha un valore maggiore rispetto a x_1.

Strategia: Partire dalla fine, quindi da quelli più significativi, e controllare **se esiste un assegnamento di verità** con quella variabile. Se esiste, sicuramente sarà tra i migliori essendo quella più significativa. Questo iterativamente fino al primo (sono polinomiali in numero). Dopo averlo controllato, esiste un assegnamento di verità per le variabili che soddisfa la formula?

Questo problema è Δ_2^P

5.7 Nucleolus e Membership

Definiamo un **problema lineare**:

- $\min_\epsilon$

- e(S,x) $\leq \epsilon_1$
- x $\in X(G)$

$$\forall S \subset N, S \notin W_0 = \{\emptyset\}$$

Non ho capito un cazzo onestamente e dalle slides non lo capisco. Vedremo come fare, dai..

6 Giochi con interazione ristretta

Pensiamo ad un problema di grafi ma con dei vincoli che non permettono delle coalizioni tra giocatori.

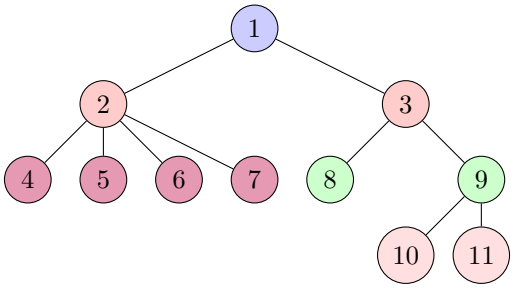


Figure 3: Esempio di albero

Pensiamola cosi. Gli agenti sono ordinati su una linea.

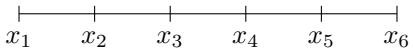


Figure 4: Esempio di linea

Il numero di coalizioni massimo che possiamo formare è **polinomiale** $\mathcal{O}(n^2)$.
E se invece di una linea avessimo un **ciclo?**

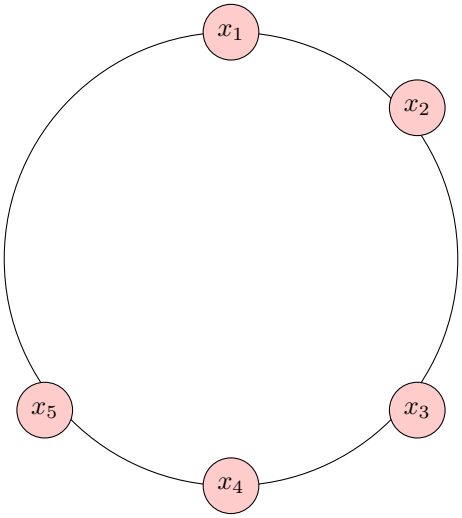


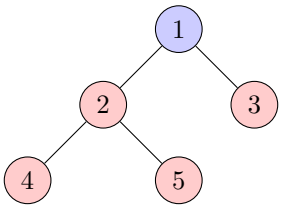
Figure 5: Esempio di ciclo

Dobbiamo contare il numero di coalizioni. Se cancelliamo l'arco x_1x_5, diventa una linea. Questo porta ad una linea, allora ancora $\mathcal{O}(n^2)$.

La **risposta** è: $\mathcal{O}(n^3)$ dato da: $n * n^2$.

E se invece avessimo **gli alberi**?

A quanto pare sugli alberi il risultato è **esponenziale**.



7 Giochi Competitivi o Non Cooperativi

I giochi competitivi o non cooperativi sono i più studiati. Si assume che un numero di agenti agisca strategicamente e sia interessato solo al proprio guadagno, senza collaborare. Cosa succede quando ogni agente ha un obiettivo e alcuni interessi, e forse alcuni di essi sono in contrasto?

Abbiamo un insieme di agenti che chiamiamo N (per esempio $N = \{1, 2, \ldots\}$). Ognuno di essi ha alcune azioni possibili tra cui scegliere, quindi abbiamo un insieme di azioni A_i che è un sottoinsieme di tutte le possibili azioni $A_i \subset A$ (per esempio $A_i = \{p, q, \ldots\}$). Quando scegliamo un'azione, otteniamo un valore per questa azione, e c'è un premio associato che è un numero reale. Questo premio dipende non solo dalle nostre azioni ma anche dalle azioni degli altri giocatori, quindi il valore dipende da noi e dagli altri agenti.

Ciascun agente è associato a un'utilità u_i che dipende da tutte le strategie:

$$u_i : A_1 \times A_2 \times A_3 \times \ldots \times A_n \to \mathbb{R}$$

Il nostro obiettivo è massimizzare la nostra utilità, ma non possiamo ottimizzare completamente poiché non possiamo controllare le variabili degli altri giocatori. Dobbiamo anche considerare cosa giocheranno gli altri giocatori.

Consideriamo un gioco con due giocatori, Bob e John. Le azioni disponibili sono $\{HOME, OUT\}$. Cosa succede quando eseguono queste azioni? Di solito, mettiamo l'utilità di entrambi gli agenti in una matrice:

	HOME	OUT
HOME	$(1, 2)$	$(1, 0)$
OUT	$(2, 0)$	$(-1, 3)$

In questo esempio, i numeri rappresentano le preferenze mappate su numeri reali.

Cosa significa questo? Sono una mappatura delle preferenze in numeri reali. Quale sarebbe l'esito? Di solito non è facile capire cosa accadrà.

	HOME	OUT
HOME	$(2, 2)$	$(1, 1)$
OUT	$(1, 0)$	$(0, 0)$

In questo esempio, l'equilibrio di Nash è {HOME, HOME}, poiché questo esito è fortemente preferito tra gli scenari. Ma cosa succede in questo caso?

	HOME	OUT
HOME	$(2, 2)$	$(1, 3)$
OUT	$(1, 0)$	$(0, 0)$

(1,3) è un equilibrio di Nash. (2,0) è un equilibrio di Nash.

7.1 Equilibrio di Nash in Giochi Puri

Una strategia s_i per un agente è un'azione, in cui scegliamo una delle azioni disponibili. Un profilo di strategia congiunta o σ è una strategia per ciascun agente del gioco. σ è un equilibrio di Nash se, per ciascun agente possibile, l'utilità dell'agente ($u(\sigma)$) ottenuta per una particolare strategia che decide di giocare dovrebbe essere almeno uguale o maggiore dell'utilità che potrebbe ottenere se gli altri giocassero la stessa strategia, ma lui prova una nuova strategia. In altre parole, giocando una strategia diversa mentre gli altri mantengono le stesse strategie, dovrebbe ottenere lo stesso valore o un valore maggiore. $\sigma = (\sigma_i$ che è s_i, σ_{-i} è la strategia degli altri).

Questo tipo di equilibrio e impostazione è chiamato "puro". Un esempio può aiutare a capire il concetto:

	HOME	OUT
HOME	$(1, 2)$	$(1, 0)$
OUT	$(2, 0)$	$(-1, 3)$

In questo caso, non esiste un equilibrio di Nash, nessuno è davvero felice, e c'è una situazione ciclica. Quando ci concentriamo sull'equilibrio di Nash, possiamo avere uno, più di uno o nessuno.

Per calcolare un equilibrio di Nash, è necessario iterare tra le strategie degli agenti e tra le strategie di ciascun agente. Il limite è dominato dalla dimensione del prodotto cartesiano: $|A_1 \times A_2 \ldots \times A_n| \times |N| \times \max |A_i| \geq 2^n$. Ciò rende difficile rappresentare matrici quando il numero di agenti è grande, poiché la dimensione della rappresentazione è esponenziale rispetto al numero di agenti.

7.2 Equilibrio di Nash in Giochi Grafici

Ma cosa succede con l'equilibrio di Nash in giochi non puri? Supponiamo di avere un grande gioco di popolazione, interagirai davvero con tutti i partecipanti al gioco? Invece, interagirai con i tuoi vicini (amici e nemici), quindi creerai un grafo con nodi che corrispondono esattamente agli agenti. L'utilità dipende dalle tue azioni e dalle azioni dei tuoi vicini. Questo è noto come "giochi grafici". I tuoi vicini sono un sottoinsieme di N e sono i nodi adiacenti. La funzione di utilità è $u_i : A_i \times \prod_{j \in \text{neigh}_i} A_j \to \mathbb{R}$. Se hai al massimo k agenti intorno a te, allora sappiamo che la dimensione (la matrice, l'utilità) è di circa $O(2^k)$, ma poiché k è limitato, questa rappresentazione è ragionevole.

Tuttavia, è importante notare che anche se non sei direttamente connesso, gli altri influenzano comunque i tuoi vicini, quindi alla fine è necessario considerare tutti i giochi e tutti gli agenti.

7.3 Complessità Computazionale

Consideriamo il problema: esiste un equilibrio di Nash in un gioco grafico (IMPOSTAZIONE PURA)? Siamo in NP? - Dobbiamo indovinare un profilo di

strategia, il che richiede tempo polinomiale, quindi la dimensione di σ è polinomiale rispetto alle dimensioni dei giochi. - Verifica se il profilo di strategia è un equilibrio di Nash: dobbiamo iterare su tutti gli agenti possibili e su tutte le possibili azioni $|N| \times |A_i|$, il che è anch'esso polinomiale.

Quindi sappiamo che questo problema è in NP. Ma è anche NP-hard?

Supponiamo di avere due agenti e tre azioni:

	R	G	B
R	$(0,0)$	$(1,1)$	$(1,1)$
G	$(1,1)$	$(0,0)$	$(1,1)$
B	$(1,1)$	$(1,1)$	$(0,0)$

7.4 Esempio di Equilibrio di Nash

In questo esempio, cerchiamo di capire come ottenere un equilibrio di Nash in un gioco usando un esempio di colorazione di grafo. Supponiamo che vogliamo che due agenti, a e b, siano felici solo se scelgono colori diversi. Abbiamo tre colori: Rosso (R), Verde (G) e Blu (B).

		a				
		R		G		B
a	R	$(0,0)$	$(1,1)$	$(1,1)$	$(1/2,1/2)$	
	G	$(1,1)$	$(0,0)$	$(1,1)$		
	B	$(1,1)$	$(1,1)$	$(0,0)$		
	U					$(2,2)$
		R		G		B
		b				

In questo modo, possiamo sempre far sì che gli agenti siano felici quando scelgono colori diversi. Supponiamo che il grafo sia 3-colorabile, il che significa che è possibile colorare i nodi in modo che nessun nodo adiacente abbia lo stesso colore. In questo caso, esiste un equilibrio di Nash.

Supponiamo ora che il grafo non sia 3-colorabile, il che significa che ci saranno sempre due nodi che sceglieranno lo stesso colore, ma per uno di loro non sarà conveniente cambiarlo. Questo "ingranaggio" potrebbe non funzionare, ma potrebbe sempre esistere un equilibrio di Nash.

Inoltre, aggiungiamo un'ulteriore variabile, U, che rappresenta la situazione in cui non è possibile ottenere un colore diverso e quindi giochiamo U. In questa situazione, se un vicino sta giocando U, vogliamo anche noi giocare U. Ma in questo modo potremmo finire in un equilibrio di Nash in cui tutti giocano U, il che non vogliamo. Per affrontare questa situazione, aggiungiamo un giocatore fittizio, detto anche "dummy player", che non è felice a giocare U, ma si preoccupa solo se qualcun altro sta giocando U. Abbiamo quindi due giocatori, α e β, che giocheranno come segue:

	α	β
α	$(1,-1)$	$(-1,1)$
β	$(-1,1)$	$(1,-1)$

In questo modo, otteniamo un'utilità che non scatena una situazione di soddisfacibilità. Pertanto, assumendo che il grafo non sia 3-colorabile, potrebbe non esistere un equilibrio di Nash.

8 Metodi di decomposizione strutturale e Isole di tracciability per problemi NP-Hard

Stiamo partendo da questo grafico per quanto riguarda il **Kakutani's fixed point theorem**:

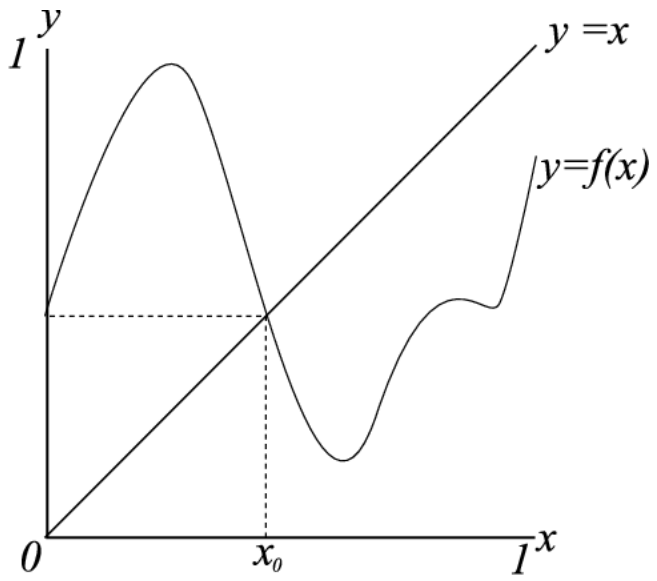


Figure 6:

dove sull'asse delle x abbiamo la **probabilità di fare un'azione**, e sull'asse delle y la nuova probabilità di eseguire quell'azione.

$$x \to y$$

ma x e y sono la stessa cosa. E' una funzione che va in se stessa.

Il momento in cui si interseca la bisezione del quadrante, sei sicuro che **quel punto esiste** e che quel punto è un **equilibrio di Nash**.

Non ho la minima idea di cosa stia succedendo qui, ma vediamo cosa ha da dire chat.

Risposta di Copilot Chat:

Nella teoria dei giochi, il teorema del punto fisso di Kakutani è usato per dimostrare l'esistenza dell'equilibrio di Nash. Un equilibrio di Nash è una situazione in cui nessun giocatore può migliorare il proprio risultato cambiando

unilateralmente la propria strategia, dato che le strategie degli altri giocatori rimangono invariate.

La connessione tra il teorema del punto fisso di Kakutani e l'equilibrio di Nash è la seguente: si può pensare a ogni strategia come a un punto in uno spazio, e a ogni risposta ottimale a una strategia come a un altro punto in quello spazio. L'equilibrio di Nash corrisponde a un punto fisso in questo spazio: un punto che è mappato su se stesso dalla funzione che assegna a ogni strategia la sua risposta ottimale.

In altre parole, l'equilibrio di Nash è una situazione in cui la "migliore risposta" di ogni giocatore alla strategia degli altri è semplicemente la strategia che sta già giocando. Questo è esattamente il tipo di situazione che il teorema del punto fisso di Kakutani è progettato per gestire, quindi può essere usato per dimostrare che un tale equilibrio deve esistere.

Nei giochi **puri**, *trovare* un equilibro di nash è **NP-Complete**. E nel caso dei giochi **misti**, in quale complessità siamo?

Nei giochi misti sappiamo subito che la risposta è **si!**, ma dalla *proof* si vede che non è facile trovare il valore di f. In letteratura, si è visto che in questi casi si può ottenere un equilibro di nash con valore *irrazionale*, come $\frac{1}{\sqrt{2}}$.

$$\begin{gathered} a_i, b_i, c_i \\ \text{utility} = f(\sum a_i \times b_i \times c_i) \\ \text{Si può ottenere } \sqrt{3} \end{gathered} \tag{4}$$

E alla lavagna ha scritto, rispettivamente come per i giochi puri **np-complete**, per i giochi misti ha scritto **computazione**.

8.1 PPAD

(Polynomial Parity Argument on Directed Graphs)

C'è un grafo particolare per questa complessita, che è il seguente:

A quanto pare navigare questo grafo era importante e rappresentava qualcosa, ma purtroppo non ho capito perché è letteralmente andato come un razzo e non c'è assolutamenteniente su internet di quello che diceva Greco.

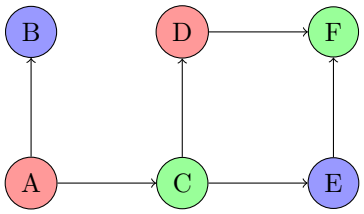
8.2 Tree Decomposition

Partiamo a parlare di complessità esponenziale. L'esempio principale è quello del **domino**. Il domino è un gioco che si gioca con delle tessere, che hanno due numeri da 0 a 6, e si devono mettere in fila in modo che i numeri combacino. Il gioco è finito quando non si possono più mettere tessere in fila.

In generale, dato un insieme di tessere, siamo in grado di mettere in uno spazio limitato? Si, è NP-Hard: si guessa e si controlla. Ma se vogliamo fare la stessa cosa per uno spazio senza limiti, come facciamo? Il problema è esponenziale.

Esempio 8.1 *(3 colorabilità)*

- Input: Un grafo G
- Domanda: Il grafo G è 3-colorabile?

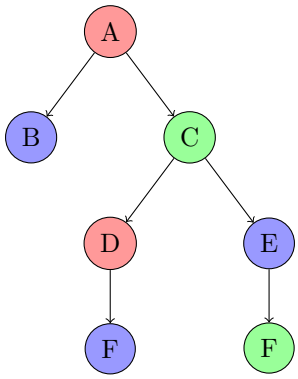


Parola chiave: **i cicli**. A quanto pare i cicli sono il **male** in informatica.

Il problema della 3 colorabiltà è **NP-Hard**, ma se abbiamo controllo sui grafi e sui cicli, il problema cambia. Dobbiamo creare una **metrica** che ci permetta di gestire questa cosa: **gradi di ciclicità di un grafo** (In inglese, *degree of cyclicity*)

Domanda 8.1 *(Come definiamo un grafo senza cicli) Un albero è una struttura dati che può essere definita un grafo senza cicli.*

Banalmente, *se vediamo il grafo come un albero*, il problema della 3 colorabilità diventa **triviale**. **Non c'è bisogno di modificare le scelte precedenti**, perché per ogni nodo sai quale colore scegliere per i figli; non c'è più bisogno di un algoritmo di backtracking.



Ad esempio, dato un grafo, trovare il numero di vertici per rendere il grafo aciclico.

8.2.1 Proposta 1: Feedback Vertex Number

Feedback Vertex Number

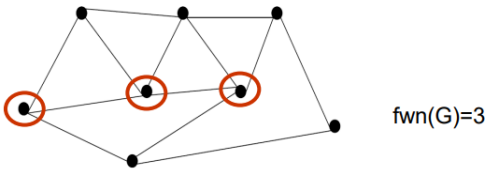


Figure 7: Feedback Vertex Number

In questo caso k è fissato.

Ma effettivamente, **è una buona misura** quella di calcoalre quanti vertici rimuovere come **grado di aciclicità**?

- Pro: Per k fissato, possiamo controllare in tempo quadratico $fwn(G) = k$
- **Contro**: Grafi semplici possono avere valori FVN grandi

8.2.2 Proposta 2: Feedback Edge Number

Una misura che ci dice **quanti archi rimuovere per rendere il grafo aciclico**.

A quanto pare, il problema rimane lo stesso.

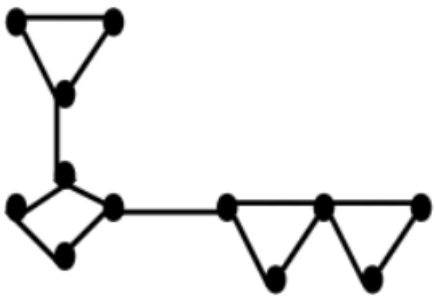


Figure 8: Feedback Edge Number

La soluzione a questi problemi è quella di **gruppare** i componenti e creare un cluster.

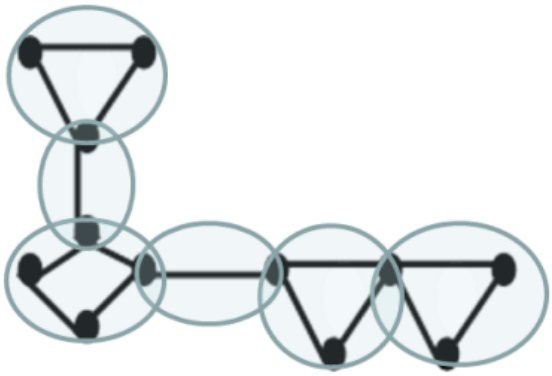


Figure 9: Biconnected Component

8.2.3 Biconnected Width

Definizione 8.1 *(Biconnected Component) Una componente biconnessa è un sottografo massimale che non contiene* ***punti di articolazione****, cioè un grafo che rimane connesso se rimuovo un nodo del grafo.*

- Pro: bcg(G) può essere calcolato in tempo lineare
- **Contro**: Aggiungere un **singolo arco** ha un effetto tremendo al bcw(g)

8.2.4 Deep dive nella tree decomposition

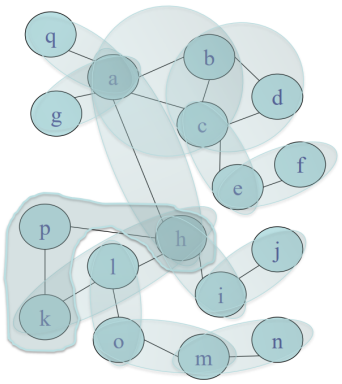


Figure 10: Grafo G1

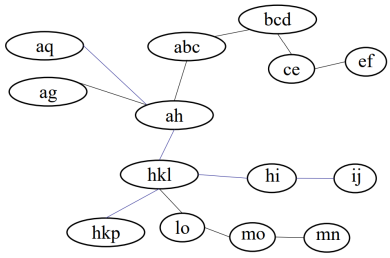


Figure 11: Grafo G2

Entrambe sono la stessa cosa, ma la prima è più **compatta** della seconda. La seconda è più **esplicita**.

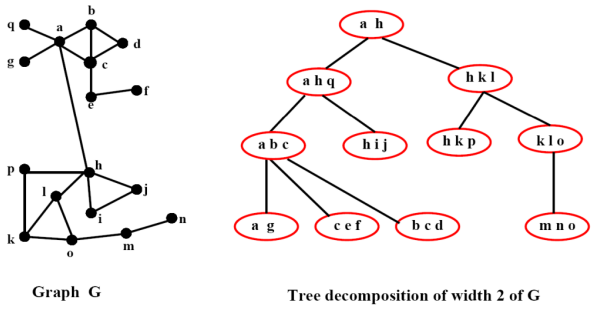


Figure 12: Tree Decomposition con width 2

Praticamente, per ogni nodo, si crea un **bag** che contiene i nodi d **vicini** e i nodi **figli**.

Quali sono le proprietà da rispettare?

- Ogni arco deve essere **coperto**
- Se prendo un vertice qualsiasi, il sotto grafo deve rimanere connesso

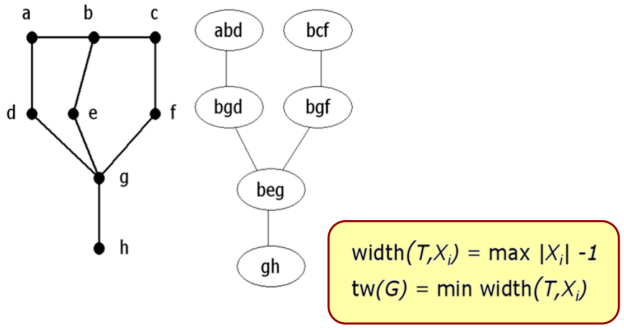


Figure 13: Tree Decomposition con width 2

In questo caso di figura 13 le proprietà sono rispettate!

Tree width: *min width* su tutte le possibili composizioni.

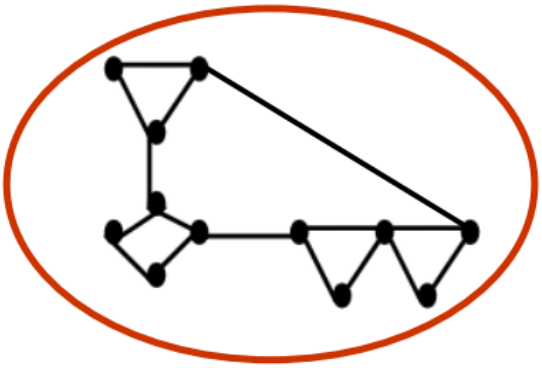
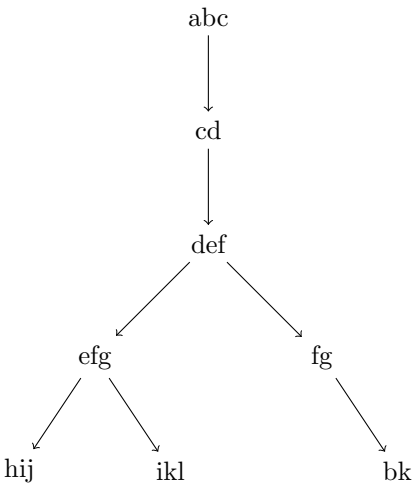
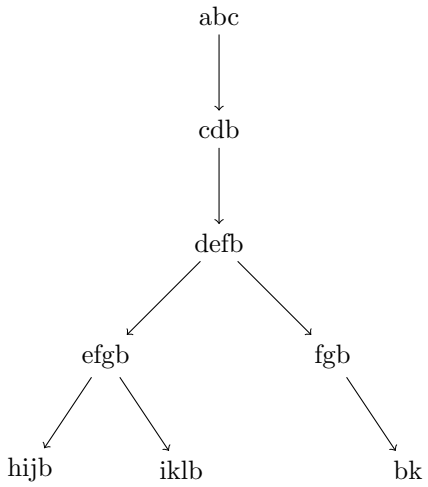
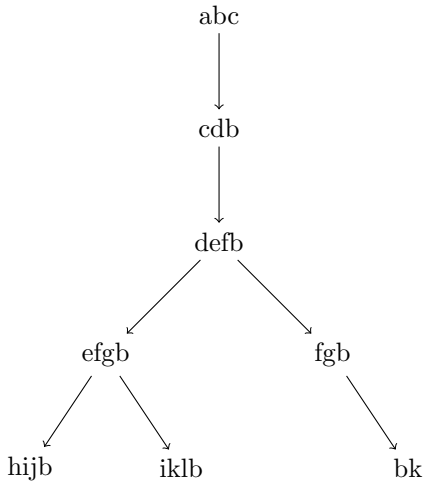


Figure 14: Tree Width in caso particolare

In questo caso, cerchiamo di capire come calcolare la *tree width*.





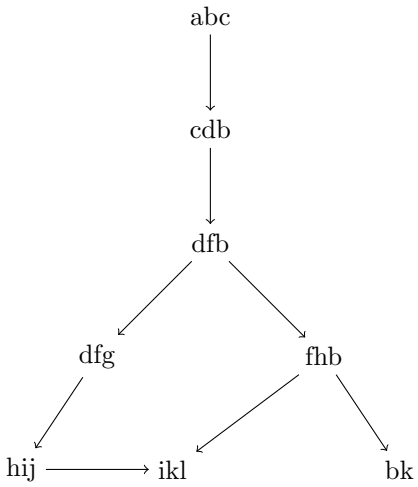


Ma possiamo ottenere un valore migliore? Possiamo ottenere 2?

La risposta è **si**, con questa configurazione posso ottenere una width di 2.

Le domande sono 2 ora: **Come si calcola una decomposizione?** e l'altra è **Quando ho una decomposizione, come si risolve un problema NP-Hard?**

Corollario 8.0.1 *(Correlazoine NP-Hard e Tree Width) Tutti i problemi NP-Hard* $2^n \to 2^k$*, cioè avendo* $k =$ ***tree width****, i problemi NP-Hard sono risolvibili in tempo* 2^k*.*



8.2.5 Robber and Cops Game

Definizione 8.2 *Il gioco Robber and Cops è un gioco dove un* ***Robber*** *deve essere catturato da un gruppo di* ***Cops****, e i Cops possono bloccare le strade per catturare il Robber.*

- *Input: Un grafo G e un intero k*

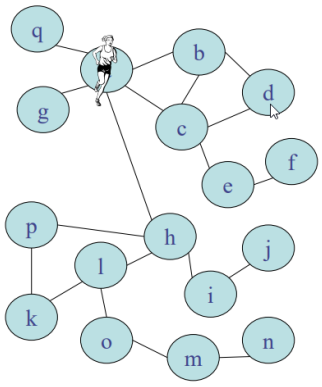


Figure 15: Cop1

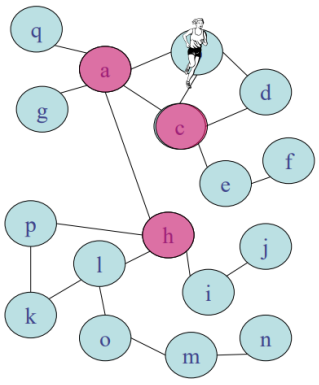


Figure 16: Cop2

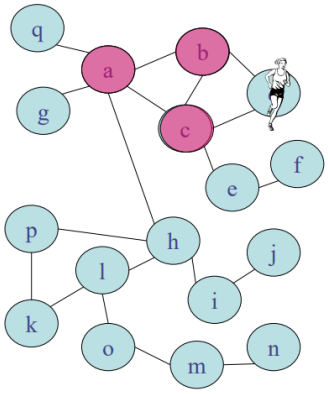


Figure 17: Cop3

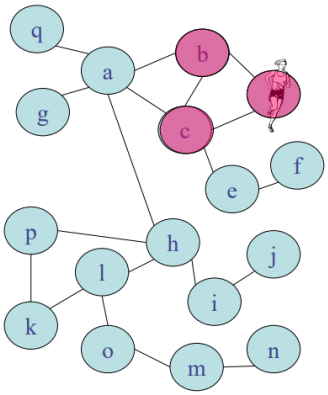


Figure 18: Cop4

Cosa capiamo da questo? Che cercare di **vedere le possibili mosse** da fare per bloccare un ladro nel grafo, praticamente ci tira fuori una **struttura** che è una **tree decomposition.**

Nota: quante combinazioni di stati iniziali ci sono?

$$n^k$$

dove n è il numero di nodi e k è il numero di poliziotti.

Nota 2: Il caso peggiore per una tree decomposition è la **cricca**.

Definizione 8.3 *(Cricca) Una cricca è un sottografo completo, cioè un grafo in cui ogni nodo è collegato a tutti gli altri nodi.*

Un approfondimento algoritmico sulla **tree decomposition** dovrebbe essere disponibile alla lezione di **laboratorio** 12.6.

♦ 8.2.6 3 Colorabiltà con Tree Decomposition

Abbiamo un albero con questi nodi:

- yp
- yzv
  - zvw
  - vz

Per ogni nodo, prende esattamente 3^k per risolvere la 3 colorabilità.

- yp:
- – RB

  - RG
  - BG
  - BR
  - GR
  - GB

- yzv:
  - RBG
  - RGB
  - BRG
  - BGR
  - GRB
  - GBR

- zvw:
  - RBG
  - RGB
  - BRG
  - BGR
  - GRB
  - GBR

- vz:
  - * RB
    * RG
    * BG
    * BR
    * GR
    * GB

Dove vuole arrivare? Vuole dire che se calcoli le soluzioni per i sotto problemi, si nota che ci sono delle soluzioni che vanno bene anche per alcuni problemi più grandi. Quindi si calcolano le **soluzioni locali** che sono 2^k e si utilizzano queste soluzioni per risolvere gli altri. E' quindi praticamente un **algoritmo di programmazione dinamica**.

9 Oltre la Tree Decomposition

L'ultima volta abbiamo visto algoritmi che avevano un k fissato, avendo complessità $\mathcal{O}(n^k)$. Ora vediamo un altro caso.

$$\mathcal{O}(n^\alpha \cdot f(k)) = \mathcal{O}(n \cdot 2^k)$$

Questi due sono rispettivamente: **Genuine Tractability** e **Fixed Parameter Tractability**.

Nota: Ricordiamo che k è la width.

9.1 Monadic Second Order Logic

Teorema 9.1 *(Teorema di Courcelle) Sia P un problema sui grafi che può essere formulato in **Monadic Second Order Logic** (MSO). Allora, il problema P può essere risolto in tempo lineare sui grafi con **treewidth** fissata.*

Esempio 9.1 *(3 colorabilità in MSO) pag 53*

***Monadic**: I predicati hanno una variabile. Sono unari.*

***Una sola relazione**: Si risolve il problema con una sola relazione. In questo caso, la relazione $E(x, y)$.*

Domanda: Quanto è difficile impostare il problema come un grafo? E' difficile?

Pensiamo a $SAT : \exists x_1 x_2, \dots, x_n (x_1 \vee x_2 \vee \dots \vee x_n), \dots (x_1 \vee \neg x_2 \vee \dots \vee \neg x_n)$. Come lo portiamo ad un grafo?

Proposta 1: Con una **reduction**, ma possiamo in un altro modo.

Consideriamo le *clausuole* cioè $(x_1, \dots \neg x_n)$ tipo. Per ogni clausola, creiamo un nodo. Per ogni letterale, creiamo un nodo.

Ora, consideriamo 2 relazioni $edge_p(x, y)$ e $edge_n(x, y)$ che indicano valori **positivo** e **negativo**.

$$\exists T \, \forall x \; s.t. \; [(edge_p(x, \_) \vee edge_n(x, \_))] \implies \exists y edge_p(x, y) \wedge T(y) \vee edge_n(x, y) \wedge \neg T(y)$$

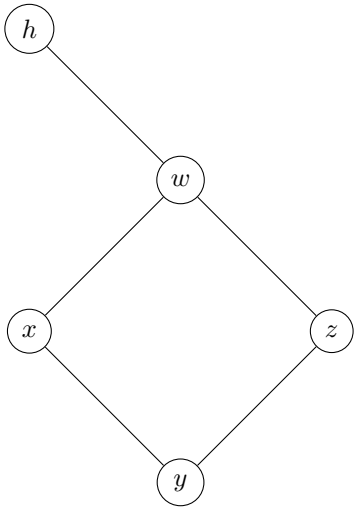
9.2 Query Evaluation

Quanto è complesso controllare che una query sia valida? Dipende se la query è **fissata**. Vediamo un'esempio con la **3 colorabilità**.

Esempio 9.2 *(3-COL con Query Evaluation (BCQ-EVAL))*

$$\exists x, y, z, w, h r(x, y) \wedge r(y, z) \wedge r(x, w) \wedge r(z, w) \wedge r(w, h) \tag{5}$$

E considerando un grafo



r	**-**
R	G
R	B
G	B
...	...

Ci sono alcuni grafi semplici che sono molto ciclici. Ad esempio, le **cricche.** Alcuni problemi sono meglio descritti da alcune strutture dati che prendono il nome di **iper grafi**.

Consideriamo una relazione $r(x_1, x_2, \ldots, x_n)$. Quanto è la treewidth? $n-1$.

Ha detto molta roba qui, ma non ho proprio capito.

9.3 Iper grafi

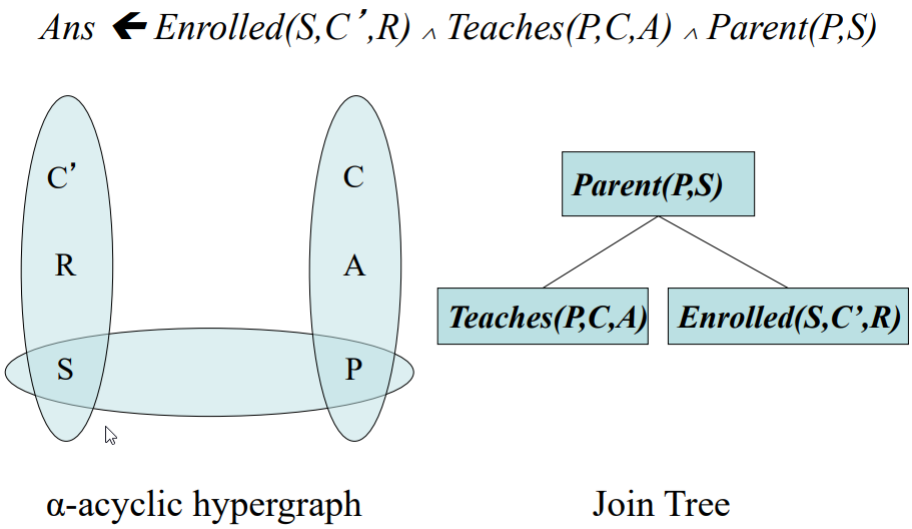


Figure 19: Ipergrafo aciclico

ONestamente, di quest parte non ho capito niente e NON VOGLIO CAPIRCI NIENTE. Non si capisce niente.

Esempio 9.3 *(Applicazione di Gioco Succinto)*
A pagina 206 se vuoi leggere e provare a capire

9.4 Problemi di Ottimizzazione

Appunti di Laboratorio

10 Introduzione alla teoria dell'utilità e decision making

10.1 Concetti di base

10.1.1 ALTERNATIVE

Parliamo di *agenti* che devono scegliere un'*alternativa* da un'insieme $\mathcal{X}$ di alternative. Questo insieme di alternative ha degli elementi che possono essere **esaustivi** o **mutualmente esclusivi**.

Esepmio: § {

- DL = Deep Learning
- AGT = Algorithmic Game Theory
- DLAGT = Deep Learning Algorithmic Game Theory
- N = None

}

10.1.2 PREFERENZE

Con il termine **preferenze** identifichiamo una relazione $\succcurlyeq$ su $\mathcal{X}$, che è un sottoinsieme di $\mathcal{X} \times \mathcal{X}$. Le preferenze possono essere:

- **complete** se $\forall x, y \in \mathcal{X}$ vale $x \succcurlyeq y$ oppure $y \succcurlyeq x$
- **transitive** se $\forall x, y, z \in \mathcal{X}$ vale $x \succcurlyeq y$ e $y \succcurlyeq z$ allora $x \succcurlyeq z$

10.1.3 Relazione di Preferenza

Una preferenze è una **relazione di preferenza** se è sia **completa** che **transitiva**.

Si chiama preferenza **stretta** se $x \succ y \iff x \succcurlyeq y$ e $x \not\succcurlyeq y$.
Si chiama **indifferenza** se $x \sim y \iff x \succcurlyeq y$ e $x \preccurlyeq y$.

10.1.4 Rappresentare le preferenze come utilità

Una relazione di preferenza puà essere tradotta in una funzione di utilità del tipo $u : \mathcal{X} \to \mathbb{R}$. Si può fare in questo modo:

$$x \succcurlyeq y \iff u(x) \geq u(y) \quad \forall x, y \in \mathcal{X} \tag{6}$$

Esempio: Se un agente dovesse trovare x almeno buono quanto y, allora la funzione di utilità $u(x)$ deve essere almeno alta quanto $u(y)$. Cioè l'agente è come se stesse **massimizzando** il valore di $u(var)$.

Teorema 10.1 (Rappresentazione Ordinale) *Sia $\mathcal{X}$ un insieme finito di alternative e sia $\succcurlyeq$ una relazione di preferenza su $\mathcal{X}$. Allora una preferenza può essere rappresentata come una funzione di utilità $u : \mathcal{X} \to \mathbb{R}$ se e solo se è **completa** e **transitiva**. In più, se $f : \mathcal{R} \to \mathcal{R}$ è una funzione monotona crescente, allora $f \circ u$ rappresenta la stessa preferenza di $u \succcurlyeq$*

***Nota**: dall'ultimo statement, l'ordine ha rilevanza.*
Per essere valido ci sono 2 condizioni necessarie:

- *Transitività: Cioè, dato $\mathcal{X} = \{a, b, c\}$, suponiamo che $a \succ b \succ c \succ a \implies u(a) > u(b) > u(c) > u(a)$. Questo sarebbe **assurdo**.*
- *Completezza: Se abbiamo preferenze incomplete, allora al massimo possiamo costruire un ordine per un sottoinsieme di $\mathcal{X}$.*

Dimostrazione 10.1 (Rappresentazione Ordinale)

La transitività e la completezza sono necessarie e sufficienti. Supponiamo di avere l'insieme $X = \{X_1, \ldots, X_n\}$. Possiamo suddividere gli elementi di X in k classi di indifferenza $C_1, \ldots, C_k$ tali che $C_1 \succ C_2 \succ \ldots \succ C_k$. In questo modo, possiamo definire la funzione di utilità u in modo che:

$$\begin{aligned} u(x) &= k \quad \forall x \in C_1, \\ u(x) &= k - 1 \quad \forall x \in C_2, \\ &\ldots \\ u(x) &= 1 \quad \forall x \in C_k. \end{aligned}$$

In questo contesto, $\succ$ rappresenta la relazione di preferenza.

10.2 Le lotterie

Una lotteria è una tupla $\mathcal{L} = (p_1, x_1; p_2, x_2 \ldots, p_n, x_n)$.

- Con prezzo monetario $x_1, x_2, \ldots, x_n \in X \subseteq R$.
- Distribuzione di probabilità $(p_1, p_2, \ldots, p_n)$.

Quindi con $\mathcal{L}$ viene definito l'insieme delle lotterie semplici.
Un esempio di lotteria è il seguente:

$$L = (0.3, 10; 0.2, 5; 0.1, 0; 0.4, -5)$$

Possiamo calcolare il valore atteso per la lotteria come:

$$\mathbb{E}(L) = \sum_{i=1}^{n} p_i x_i$$

$$\mathbb{E} = 0.3 \times 10 + 0.2 \times 5 + 0.1 \times 0 + 0.4 \times (-5) = 2$$

Definizione 10.1 (Utilità attesa) *Data una relazione di preferenza $\succcurlyeq$ su $\mathcal{L}$, una funzione di utilità $U : \mathcal{L} \to \mathbb{R}$ è funzione di utilità attesa se può ssere scritta come:*

$$U(L) = \sum_{i=1}^{n} p_i u(x_i)$$

dove p_i è la probabilità che l'evento x_i accada e u è la funzione di utilità di Bernoulli.

per un qualche funzione $u : R \to R$.

Questa funzione u viene chiamata **funzione di utilità di Bernoulli**

10.3 Utilità di Von Neumann-Morgenstern

I due matematici Von Neumann e Morgenstern hanno dimostrato che se una relazione di preferenza $\succcurlyeq$ su $\mathcal{L}$ soddisfa le seguenti proprietà, allora può essere rappresentata come una funzione di utilità:

- **Assioma 1** (Ordine di preferenza): $\succcurlyeq$ è **completa** e **transitiva**
- **Assioma 2** (Continuità): Se $L \succ M \succ N$ allora esiste $p \in [0, 1]$ tale che $pL + (1 - p)N \sim M$
- **Assioma 3** (Indipendenza): Per una qualsiasi lotteria N e $p \in [0, 1], L \succcurlyeq M \iff pL + (1 - p)N \succcurlyeq pM + (1 - p)N$

Teorema 10.2 (Teorema di VNM) *Una relazione binaria $\succ$ su $\mathcal{L}$ ha una rappresentazione utilità attesa se e solo se soddisfa gli assiomi da 1 a 3. Ancora, se U e V sono rappresentazioni di utilità attesa di $\succ$, allora esistono delle costanti $a, b \in \mathbb{R}$ tale che $U(\cdot) = aV(\cdot) + b$.*

Che detto in parole italiane, significa che se una relazione binaria è completa, transitiva, continua e indipendente, allora può essere rappresentata come una funzione di utilità attesa.

Andremo a fare la **dimostrazione** in due fasi:

1. Dimostriamo che $U(L) \sum_{i=1}^{n} p_i u(x_i)$
2. Dimostriamo che $L \succ M \iff U(L) \succ U(M)$ con $L, M \in \mathcal{L}$

Dimostrazione 10.2 (Parte 1 VNM)

Supponiamo di avere n risultati $o_1, \ldots, o_n$.

Per la **completezza** e per la **transitività** possiamo ordinare i nostri risultati dal peggiore al migliorare

$$o_1 \preccurlyeq \cdots \preccurlyeq o_n$$

Sia $u(0_1) = 0$ e $u(o_n) = 1$.

Per ogni provabilità $p \in [0, 1]$, definiamo una lotteria $\mathcal{L}(p) = p \cdot o_n + (1-p) \cdot o_1$.

Per l'assioma di continuità c'è una probabilità $q_1 \in [0, 1]$, per ogni risultati, tale che $L(q_i) = o_i$ e $u(o_i) = q_i$

Segue che l'utilità della lotteria $\mathbb{M} = \sum_i p_i o_i$ è il valore atteso di u.

$$u(M) = u(\sum_i p_i o_i) = \sum_i p_i u(o_i) = \sum_i p_i q_i$$

Dimostrazione 10.3 (Parte 2 VNM)

Supponiamo che $L `succ M$, possiamo definire L' e M' come segue:

$$L' = U(L) \cdot o_n + (1 - U(L)) \cdot o_1$$
$$M' = U(M) \cdot o_n + (1 - U(M)) \cdot o_1$$

Abbiamo il seguente ordine: $\mathcal{L}' \sim L \succ M \sim M'$.
Poiché $L' \succ M' \implies U(L) > U(M)$.
Questo implica $L \succ M \iff U(L) > U(M)$.

10.4 Atteggiamento verso il rischio

C

Consideriamo una lotteria giusta $\mathcal{L} = p \cdot x + (1 - p) \cdot y = 0$
Allora:

- Un giocatore è **neutrale al rischio** $\iff$ la sua funzione di utilità è **lineare**. Cioè: $u(x) = ax + b$. *Si dice che un giocatore neutrale al rischio sia neutrale con le lotterie giuste*
- Un giocatore è **avverso al rischoi** $\iff$ la sua funzione di utilità è **concava**. *Un giocatore avverso al rischio non gioca a nesusuna lotteria giuste*
- Un giocatore è **propenso al rischio** $\iff$ la sua funzione di utilità è **convessa**. *Un giocatore propenso al rischio gioca a tutte le lotterie giuste*

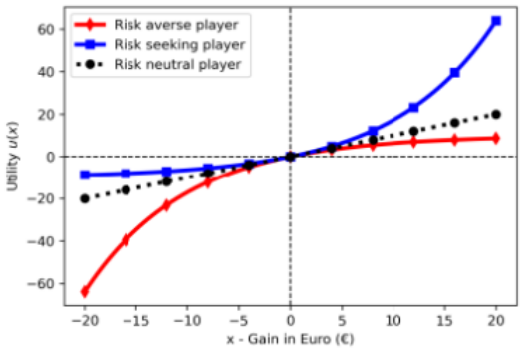


Figure 20: Rappresentazione grafica del rischio

10.5 Applicazioni: Condivisione del rischio

Immaginiamo di avere due giocatori A e B che sono *avversi al rischio* con $u(x) = \sqrt(x)$ e due *assets* $A_1, A_2 = (0.5, 100; 0.5, 0)$. Supponiamo che A_1 e A_2 siano indipendenti.

L'utilità di A_1 e A_2 è : $u(A_1)0u(A_2) = 0.5 \times \sqrt(100) = 5$

Se i due giocatori formassero un **fondo comune** dove ogni giocatore ha una quota della metà, ogni giocatore ha l'asset $A_m = (0.25, 100; 0.5, 50; 0.25, 0)$.

L'utilità di un gioatore è:

$$u(A_m) = 0.25 \times \sqrt(100) + 0.5 \times \sqrt(50) \approx 6.$$

Questo perché il giocatore ha una probabilità del 50% di avere 100 e una probabilità del 50% di avere 50. Quindi la media è 75 e la radice è 6.

10.6 Applicazione: Assicurazione

Supponiamo di avere un giocatore A che è *avverso al rischio* con $u(x) = \sqrt(x)$ e un asset $A = (0.5, 100; 0.5, 0)$.

Immaginiamo di avere una compagnia di assicurazione neutrale al rischio con tantissimi soldoni.

Quale premium P pagherebbe il giocatore per assicurare il suo asset?

$$u(100-P) \geq 0.5 \times u(100) + 0.5 \times u(0) \rightarrow \sqrt(100-P) \geq 5 \rightarrow 100-P \geq 25 \rightarrow -P \geq -100+25 \rightarrow P \geq 75$$

Quale premium pagherebbe la compagnia assicurativa per assicurarsi l'asset del giocatore?

$$P \geq 0.5 \times 100 + 0.5 \times 0 \rightarrow P \geq 50$$

Allora, entrambi guadagnerebbero soldi se la compagnia assicurasse l'asset del player per un premium $P \in [50, 75]$.

11 Teoria dei giochi coalizionali

Definizione 11.1 (Giochi coalizionali)

Un gioco coalizionale (cioè con utilità trasferibile) è una coppia del tipo $G = (N, v)$ con:

- N = {1,. . . , n} l'insieme dei giocatori
- $\nu : 2^N \to \mathbb{R}$ la funzione caratteristica

Per ogni sotto-insieme di giocatori C, $\nu(C)$ è la quantità che i membri di C possono ottenere se *lavorassero insieme*.

Definizione 11.2 (Funzione caratteristica)

La funzione caratteristica è un mapping tra *ogni coalizione* $C \subseteq N$ o il suo rispettivo valore (*cioè l'utilità*).

Esempio 11.1 (Gelati)

Insieme dei giocatori N:

- A: 6$
- B: 3$
- C: 3$

Insieme degli assets: Gelati

- 500g: 7$
- 750g: 9$
- 1000g: 11$

Ora, abbiamo i ν che sono i valori di ogni coalizione:

- Cardinalità 1: $\nu(\emptyset) = \nu(\{A\}) = \nu(\{B\}) = \nu(\{C\}) = 0$
- Cardinalità 2: $\nu(\{A, B\}) = 750; \nu(\{A, C\}) = 750; \nu(\{B, C\}) = 0$
- Cardinalità 3: $\nu(\{A, B, C\}) = 1000$

11.1 Superadditività

Un gioco a funzione di caratteristica $G(N, \nu)$ è detto **superadditivo** se soddisfa:

$$\nu(C_1 \cup C_2) \geq \nu(C_1) + \nu(C_2) \forall C_1, C_2 \subset N \text{ t.c. } C_1 \cap C_2 = \emptyset$$

Cioè, in italiano, significa che,

Definizione 11.3 (Superadditività) *Dato un gruppo di giocatori C_1 e un gruppo di giocatori C_2, se dovessero unirsi, il valore della coalizione risultante è maggiore o uguale alla somma dei valori delle due coalizioni.*

- Cardinalità 1: $\nu(\emptyset) = \nu(\{A\}) = \nu(\{B\}) = \nu(\{C\}) = 0$
- Cardinalità 2: $\nu(\{A, B\}) = 750; \nu(\{A, C\}) = 750; \nu(\{B, C\}) = 0$
- Cardinalità 3: $\nu(\{A, B, C\}) = 1000$

Prendiamo per esempio $\nu(\{A, B\})$.

$$\nu(\{A, B\}) \geq \nu(\{A\}) + \nu(\{B\}) \implies 750 \geq 0 \tag{7}$$

Questo vale anche per $\nu(\{A, C\})$:

$$\nu(\{A, C\}) \geq \nu(\{A\}) + \nu(\{C\}) \implies 750 \geq 0 \tag{8}$$

E anche per $\nu(\{B, C\})$:

$$\nu(\{B, C\}) \geq \nu(\{B\}) + \nu(\{C\}) \implies 0 \geq 0 \tag{9}$$

11.2 Core o Nucleo

Il core o nucleo è definito come:

$$Core(G) = X \text{t.c} \begin{cases} x_i \geq 0 \forall i \in N \\ \sum_{i \in N} x_i \leq \nu(N) \\ \sum_{i \in C} x_i \geq \nu(C) \forall C \subseteq N \end{cases} \tag{10}$$

11.3 Shapley Value

Lo shapleu value di un giocatore i è la contribuzione marginale media del player i su tutte le possibili coalizioni.

$$\phi(i, \nu) = \frac{1}{|N|!} \sum_{\pi \in \prod_N} \nu(B(\pi, i)) \cup \{i\} - \nu(B(\pi, i)) \tag{11}$$

con:

- $\prod_N$ è l'insieme di tutte le possibili permutazioni di N

- $B(\pi, i)$ è l'insieme di tutti i predecessori di i nella perutazione π.

Esempio 11.2 (Shapley value con giocatore A)

- Cardinalità 1: $\nu(\{A\}) = \nu(\{B\}) = \nu(\{C\}) = 0$
- Cardinalità 2: $\nu(\{A, B\}) = 750; \nu(\{A, C\}) = 750; \nu(\{B, C\}) = 0$
- Cardinalità 3: $\nu(\{A, B, C\}) = 1000$

Ora, calcoliamo le computazioni per **A**.

- $\pi_1 = (A, B, C) \implies \nu(\{A\}) - \nu(\emptyset) = 0 - 0 = 0$
- $\pi_2 = (A, C, B) \implies \nu(\{A\}) - \nu(\emptyset) = 0 - 0 = 0$
- $\pi_3 = (B, A, C) \implies \nu(\{A, B\}) - \nu(\{B\}) = 750 - 0 = 750$
- $\pi_4 = (B, C, A) \implies \nu(\{A, B, C\}) - \nu(\{B, C\}) = 1000 - 0 = 1000$
- $\pi_5 = (C, A, B) \implies \nu(\{A, C\}) - \nu(\{C\}) = 750 - 0 = 750$
- $\pi_6 = (C, B, A) \implies \nu(\{A, B, C\}) - \nu(\{B, C\}) = 1000 - 0 = 1000$

In totale, allora, abbiamo:

$$\phi(A, \nu) = \frac{1}{6}(0 + 0 + 750 + 1000 + 750 + 1000) = 583.\overline{33}$$

Nota: Lo shapley value può essere anche calcolato utilizzando questa formula:

$$\phi(i, \nu) = \frac{1}{|N|!} \sum_{\pi \in \prod_N} \nu(B(\pi, i)) \cup \{i\} - \nu(B(\pi, i))$$

12 Problemi computazionali nella teoria dei giochi coalizionali

12.1 Limitazioni dell'approccio Naive

Quando utilizziamo una funzione caratteristica per rappresentare un gioco coalizionale, il problema di trovare una soluzione approccia un **utilizzo di memoria pari a** $\mathcal{O}(2^N)$.

```
    v = {
        frozenset(['A']): 0,
        frozenset(['B']): 0,
        frozenset(['C']): 0,
        frozenset(['A', 'B']): 1,
        frozenset(['A', 'C']): 1,
        frozenset(['B', 'C']): 1,
        frozenset(['A', 'B', 'C']): 2
    }
```

Per quanto riguarda la **complessità computazionale**, siamo comunque su $\mathcal{O}(2^N)$.

```
    def is_stable(outcome, cs):
        return all(
            [
                sum([outcome[player] for player in coalition]) >=
                    cs[coalition]
                for coalition in cs
            ]
        )
```

Listing 1: Controllo che un outcome sia stabile

```
    def shapley_value(player, cs):
        player = set([player])
        N = len(max(cs, key=len))
        shapley_val = 0

        for coalition in cs:
            s = len(coalition)ma
            marginal_contribution = cs[coalition] - \ cs[coalition
    - player]

            if marginal_contributoin:
                shapley_val += ((factorial(N-S) * factorial(S-1)) /
     \ factorial(N)) * marginal_contribution
        return round(shapley_val, 10)
```

Listing 2: Shapley value

In che modo possiamo fare meglio?

12.2 Strategie per i problemi di computazione

La soluzione è quella di *concentrarsi* su una categoria specifica di goichi, che possono essere risolti **con poco utilizzo di memoria e con algoritmi che lavorano in tempo polinomiale**-

Un'altra soluzione è quella di usare alcuni **algoritmi di approssimazioe**, come ad esempio è **l'algoritmo di Montecarlo**. Questo algoritmo funziona in tempo polinomiale e nella pratica **l'approssimazione dell'errore** è molto piccola nella pratica.

Altra soluzione, è quella di utilizzare delle **rappresentazoini compatte** per la funzione caratteristica. Questo tipo di soluzione ha un impatto minimo sul consumo della memoria, ha una **grande espressività**, ovvero che può rappresentare la maggior parte dei giochi e sopratutto **lavora in tempo polinomiale.**

12.3 Gioco del'aereoporto — Airport Game

Definizione 12.1 (Airport game) *Ci sono **N** compagnie aeree. Ogni compagnia ha bisogno di una pista di atterraggio di una certa lunghezza per i loro aereo. Le compagnie, però, possono **condividere** una pista, e quindi possono unire le forze per **costruire un'unica pista** abbastanza grande per tutti, e **dividere i costi**. Come devono fare per **dividersi i costi?***

Nella definizione del problema abbiamo un insieme $N = \{1, 2, \dots, n\}$ di giocatori, ai quali ognuno ha associato un costo c_i tale che $c_1 < c_2 < \cdots < c_n$. La funzione caratteristica è indicata come:

$$\nu(S) = \max_{i \in S} c_i \quad \forall S \subseteq N$$

Cioè, il **massimo** costo tra i giocatori che fanno parte della coalizione.
Lo **shapley value** per il giocatore i in questo gioco viene dato da:

$$\Phi_i = \sum_{j=1}^{i} \frac{c_j - c_{j-1}}{n - j + 1} \forall i \in N; \quad c_0 = 0$$

Esempio 12.1 (Airport game)

Immaginiamo di avere **4** giocatori, quindi 4 compagnie aeree. I costi sono:

$$[8, 11, 13, 18]$$

Giocatore	Aggiungi 1	Aggiungi 2	Aggiungi 3	Aggiungi 4	Shapley value
Costi marginali	8	3	2	5	
Costo P1	2				2
Costo P2	2	1			3
Costo P3	2	1	1		4
Costo P4	2	1	1	5	9

Il gioco dell'aeroporto, noto anche come Airport Game, coinvolge diversi giocatori che collaborano per contribuire ai costi associati all'aggiunta di servizi all'aeroporto. Ciascun giocatore può scegliere di aggiungere una quantità specifica di servizio, ognuna associata a un costo marginale. Il valore di Shapley è una misura di quanto ciascun giocatore dovrebbe contribuire in modo equo ai costi totali, considerando il loro contributo al gioco. Questo calcolo si basa sulla cooperazione tra i giocatori e assicura una distribuzione equa dei costi totali.

12.4 Approssimazione di Montecarlo

Definizione 12.2 (Approssimazione di montercarlo) *L'approssimazione di Montecarlo è un metodo di calcolo che si basa su* ***numeri casuali*** *per ottenere un risultato approssimato.*

Nel nostro caso, supponiamo di avere una *distribuzione di probabilità* $P(X)$ e che volessimo calcolarci $P(x)$.

Idea 1: Approssimiamo $P(x)$ usando delle frequenze semplici.

Idea 2: Generiamo un campione D di grandezza M da $P(X)$ e calcoliamo $P(x)$.

$$P_D(X = x) = \frac{M_{X=x}}{M}$$

cioè, calcoliamo la probabilità che X sia uguale a x nel campione D.

Confrontiamo ora la **formula dello Shapley Value** originale:

$$\phi(i, \nu) = \frac{1}{|N|!} \sum_{\pi \in \prod_N} \nu(B(\pi, i)) \cup \{i\} - \nu(B(\pi, i))$$

con:

- π_N l'insieme di tutte le possibili permutazioni di N
- $B(\pi, i)$ L'insieme di tutti i predecessori di i nella permutazione π

Invece, la formula dello Shapley Value approssimata con la tecnica di montecarlo è:

$$\widetilde{\phi}(i, \nu) = \frac{1}{m} \sum_{\pi \in \mathcal{P}} \nu(B(\pi, i) \cup \{i\}) - \nu(B(\pi, i))$$

dove:

- $\mathcal{P} \subset \prod_N$ è il **sottoinsieme** di tutte le possibili permutazioni di N
- $B(\pi, i)$ è l'insieme di tutti i predecessori di i nella permutazione π.

Un esempio di *pseudo-codice* per l'approssimazione è il seguente:

Esempio 12.2 (Pseudo codice Shapley Value con Montecarlo)

```
#Input: v → characteristic function; m → numero di sample
def MC_Shapley(v, m):
    φ̃_i = 0  ∀i ∈ N
    for k = 1,...,m:
        π_k = permutazione casuale di N
        for i = i,...,n:
            sv = v(B(π, i) ∪ {i} − v(B(π, i)))
            φ̃_i += sv
    for k = 1,...,n:
        φ̃_i = φ̃_i / m
    return φ̃_1, φ̃_2, ..., φ̃_n
```

Lo pseudocodice rappresenta un algoritmo per calcolare i valori di Shapley approssimati mediante il metodo del campionamento Monte Carlo (MC_Shapley). L'obiettivo dell'algoritmo è stimare i valori di Shapley per un insieme di giocatori (N) basandosi su una funzione caratteristica (v) e un numero specifico di campioni (m).

L'algoritmo utilizza un processo di campionamento Monte Carlo per calcolare una stima approssimata dei valori di Shapley. Per ogni campione, vengono generate permutazioni casuali degli insiemi di giocatori (π_k) e calcolate le differenze nei valori caratteristici (sv) quando un giocatore viene aggiunto a un insieme e poi rimosso. Queste differenze vengono sommate per ogni giocatore, accumulando una stima approssimata dei loro valori di Shapley ($\widetilde{\phi}_i$).

Dopo aver completato il numero desiderato di campioni, il valore $\widetilde{\phi}_i$ per ciascun giocatore viene normalizzato dividendo per il numero di campioni (m). Alla fine, l'algoritmo restituisce le stime approssimate dei valori di Shapley per ogni giocatore.

Questo approccio è utilizzato per affrontare il calcolo dei valori di Shapley in situazioni in cui non è possibile calcolarli in modo esatto, ma è possibile ottenere una stima accurata utilizzando il campionamento Monte Carlo.

12.5 Rappresentazioni compatte

Lo scopo delle **rappresentazioni compatte** è quello di ridurre l'impatto sulla memoria della funzione caratteristica, usando delle strutture a mo di **rete**.

Il valore di una coalizione non sarà più accessibili in $\mathcal{O}(1)$ come avveniva prima utilizzando il metodo Naive. Quello che si ottiene è un **tempo polinomiale**.

Avendo questa rappresentazione compatta, sfruttando la proprietà dello Shapley Value, ci permette di calcolare lo shapley value **in tempo polinomiale.**

Definizione 12.3 (Gioco del grafo indotto (ISG))

In questa rappresentazione, i **giocatori** sono dei nodi nel grafo. Gli archi sono le **coalizioni** di due giocatori. I pesi degli archi sono il **valore delal coalizione**.

Un gioco dei gioco del grafo indotto può essere espresso mediante questa funzione caratteristica:

$$v(C) = \sum_{i,j \subseteq C} w_{ij}$$

E possiamo calcolare lo shapley value per il player i nel seguente modo:

$$\phi_i = w_{ii} + \frac{1}{2} \sum_{j \in \Gamma(i)} w_{ij}$$

dove:

- w_{ii} è il peso dell'arco tra il nodo i e se stesso
- $\Gamma(i)$ è l'insieme dei vicini del nodo i

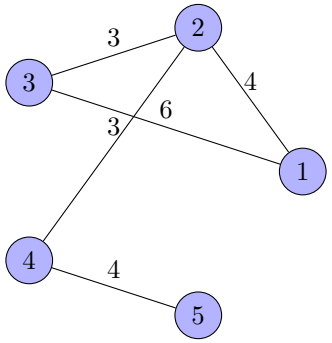


Figure 21: Esempio di grafo indotto

Ora, vediamo un paio di esempio di *coalizioni*.

Esempio 12.3 (Coalizioni nel grafo indotto)

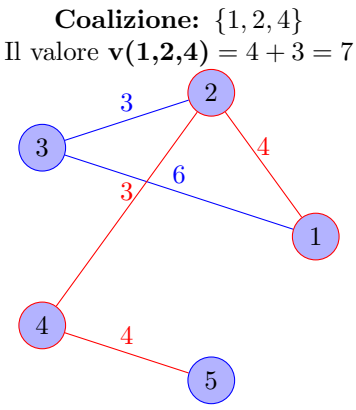


Figure 22: Esempio coalizione GF (1)

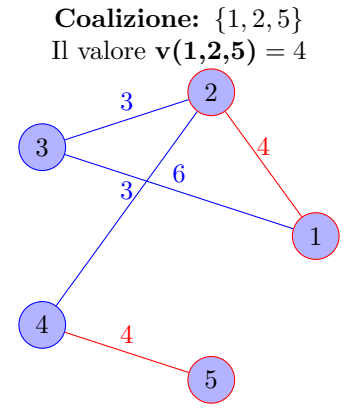


Figure 23: Esempio coalizione GF (2)

Calcoliamo ora **lo shapley value**.
Ricordiamo la formula:

$$\phi_i = w_{ii} + \frac{1}{2} \sum_{j \in \Gamma(i)} w_{ij}$$

dove:

- w_{ii} è il peso dell'arco tra il nodo i e se stesso
- $\Gamma(i)$ è l'insieme dei vicini del nodo i

Per ogni giocatore i, calcoliamo:

- $\phi_1 = \frac{1}{2}(4+6) = 5$
- $\phi_2 = \frac{1}{2}(4+3+3) = 5$
- $\phi_3 = \frac{1}{2}(6+3+4) = 6.5$
- $\phi_4 = \frac{1}{2}(4+3+4) = 5.5$
- $\phi_5 = \frac{1}{2}(4) = 2$

12.6 Reti di contribuzione marginale —MC-Nets

L'idea di questo tipo di reti è quello di rappresentare la **funzione caratteristica** come un'insieme di regole della forma:

$$pattern \rightarrow value$$

Il **pattern** è una formula booleana su N e il valore associato ad egli è il suo **contributo marginale**.

Se un pattern è nella forma $\{a \wedge b \wedge \cdots \wedge c\}$ il valore associato può essere sia *positivo* che *negativo*, possiamo **rappresentare ogni gioco.**

Definizione 12.4 (MC-Nets) *Una **rete di contribuzione marginale** è una rete che rappresenta una funzione caratteristica v come un insieme di regole della forma:*

$$pattern \rightarrow value$$

dove:

- *pattern è una formula booleana su N*
- *value è il contributo marginale*

Esempio 12.4 (MC-Nets)

$$\{a \wedge b\} \rightarrow 5$$
$$\{b\} \rightarrow 2$$

rappresenta la seguente funzione caratteristica:

$$\nu(\emptyset) = 0; \nu(\{a\}) = 0; \nu(\{b\}) = 2; \nu(\{a,b\}) = 5 + 2 = 7$$

Per capire per bene come funziona, diciamo che *dobbiamo sommare i valoriper ogni regola che si applica ad una determinata coalizione che si va a controllare.*

Una regola r per applicarsi deve essere **sotto-insieme della coalizione** che si sta andando a controllare.

Esempio 12.5 (MC-Nets coalizione {a})

Se dobbiamo controllare la coalizione $\{a\}$, vediamo quali regole si applicano. Ricordiamo il nostro insieme di regole:

$$\{a \wedge b\} \to 5$$
$$\{b\} \to 2$$

Controlliamo la prima regola: $\{a \wedge b\} \to 5$.

$$\{a, b\} \not\subseteq \{a\}$$

In questo caso, la regola **non si applica** poiché $\{a, b\}$ non è sottoinsieme di $\{a\}$.

Controlliamo la seconda regola: $\{b\} \to 2$.

$$\{b\} \not\subseteq \{a\}$$

Anche in questo caso, la regola **non viene applicata.**P

Allora, siccome dobbiamo sommare il valore di ogni regola che viene applicata, per la coalizione $\{a\}$ la somma è:

$$0 + 0 = 0$$

Quindi, assegniamo alla coalizione $\{a\}$ il valore 0.

Esempio 12.6 (MC-Nets coalizione {a,b})

Se dobbiamo controllare la coalizione $\{b\}$, vediamo quali regole si applicano. Anche qui, le regole sono le stesse:

$$\{a \wedge b\} \to 5$$
$$\{b\} \to 2$$

Controlliamo la prima regola: $\{a \wedge b\} \to 5$.

$$\{a, b\} \not\subseteq \{b\}$$

In questo caso, la regola **non si applica** poiché $\{a, b\}$ non è sottoinsieme di $\{b\}$.

Controlliamo la seconda regola: $\{b\} \to 2$.

$$\{b\} \subseteq \{b\}$$

In questo caso, la regola **si applica** poiché $\{b\}$ è sottoinsieme di $\{b\}$. Segniamo quindi il valore della regola, ovvero 2.

Allora, siccome dobbiamo sommare il valore di ogni regola che viene applicata, per la coalizione $\{b\}$ la somma è:

$$2 + 0 = 2$$

Quindi, assegniamo alla coalizione $\{b\}$ il valore 2.

Esempio 12.7 (MC-Nets coalizione {a,b})

Dobbiamo controllare l'ultimo caso. Se dobbiamo controllare la coalizione $\{a, b\}$, vediamo quali regole si applicano.

Anche qui, le regole sono le stesse:

$$\{a \wedge b\} \to 5$$
$$\{b\} \to 2$$

Controlliamo la prima regola: $\{a \wedge b\} \to 5$.

$$\{a, b\} \subseteq \{a, b\}$$

In questo caso, la regola **si applica** poiché $\{a, b\}$ è sottoinsieme di $\{a, b\}$. Segniamo quindi il valore della regola, ovvero 5.

Controlliamo la seconda regola: $\{b\} \to 2$.

$$\{b\} \subseteq \{a, b\}$$

Anche in questo caso, la regola **si applica** poiché $\{b\}$ è sottoinsieme di $\{a, b\}$. Segniamo quindi il valore della regola, ovvero 2.

Allora, siccome dobbiamo sommare il valore di ogni regola che viene applicata, per la coalizione $\{a, b\}$ la somma è:

$$5 + 2 = 7$$

Quindi, assegniamo alla coalizione $\{a, b\}$ il valore 7.

Abbiamo ottenuto quindi tutti i valori che ci servono per calcolare lo shapley value.

- {a} $\to$ 0
- {b} $\to$ 2
- {a,b} $\to$ 7

Lo **shapley value** nel caso delle $MC - Nets$ è dato da:

$$\phi_i = \sum_{\varphi \to x \in rs_i} \frac{x}{|\varphi|}$$

dove:

- x è il valore della regola $\varphi \to x$
- $|\varphi|$ è la cardinalità della regola
- rs_i è l'insieme di tutte le regole che si applicano al giocatore i

Dobbiamo, quind, prendere solo le regole che si applicano al giocatore i.
Iniziamo con il giocatore **{a}**.
L'unica regola che si applica è quella di $\{a, b\} \to 5$.
Quindi, il calcolo è:

$$\phi_a = \frac{5}{2} = 2.5 + 0$$

Passiamo al giocatore **{b}**.
Si applicano 2 regole: $\{b\} \to 2$ e $\{a, b\} \to 5$.
Quindi, il calcolo è:

$$\phi_b = \frac{2}{1} + \frac{5}{2} = 4.5$$

Abbiamo quindi calcolato **lo shapley value** per ogni giocatore.

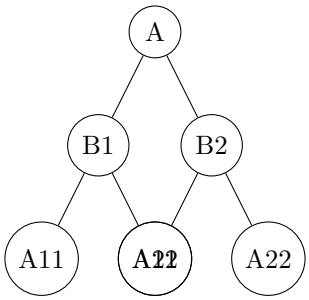
13 Non-cooperative Games

A differenza dei giochi cooperativi, in questo tipo di giochi i giocatori vogliono **massimizzare** la propria **payoff** e competono tra loro, senza collaborare.

Ci sono vari modi per rappresentarea un gioco non cooperativo

Forma normale: Ovvero in forma *matriciale*, che mostra come ogni player ottiene la propria **payoff** in funzione delle azioni. *Come se i giocatori si muovessero simultaneamente.*

Nota: Rappresentare un gioco come un albero.



Definizione 13.1 *(Gioco in Forma Normale)*

Un gioco in forma normale è identificato da una tupla $< N, A, u$ dove:

- $N = \{1, 2, \ldots, n\}$ giocatori
- $A = A_1, A_2, \ldots, A_n$ insieme delle azioni disponibili per ogni giocatore
- $u = (u_1, u_2, \ldots, u_n)$ dove $u_i : A \to \mathbb{R}$ è la funzione di payoff del giocatore i

13.1 Giochi strategia puri

In questo caso, assumiamo che *il giocatore i conosca cosa gli altri giocatori giocheranno.*

$$a_i =< a_1, a_2, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n >$$

La **strategia migliore** per a_i è quelal che massimizza il payoff:

$$a_i^* = FORMULADACOPIARE$$

Definizione 13.2 *(Equilibro di Nash puro) Un profilo di azioni è un* ***Equilibro di Nash puro*** *se ogni giocatore giocasse la migliore azoine*

$$a =< a_1, a_2, \ldots, a_n >= NE \iff \forall i a_i \in BR(a_i)$$

13.1.1 Dilemma del prigioniero

Idea del gioco: Ogni giocatore ha un incentivo di scegliere una **soluzione sub-globale ottima**.

	C	D
C	3,3	0,5
D	5,0	1,1

Spiegazione: Se entrambi i giocatori scelgono C, ottengono un payoff di 3. Se entrambi scelgono D, ottengono un payoff di 1. Se uno sceglie C e l'altro D, il primo ottiene 0 e il secondo 5. Ci sono molte strategie in gioci del genere, ad esempio tradire sempre, tradire al primo tradimento (che sembra la più funzionale).

13.1.2 Pure Coordination

Non ho capito: tipo se fai incidente non vieni pagato sennò si.

	C	D
C	a,a	0,0
D	0,0	a,a

13.1.3 Battle of the Sexes

Si ha una coppia di partner che vogliono vedere un film insieme. Si vede un film Comedy o Action. Uno dei due vuole vedere Action, uno vuole vedere Comedy. Se si vede il film da soli, non si ottiene una payoff. C'è una componente egoistica:

ognuno vuole vedere qualcosa che piace a lui. In base a se si sceglie A o C, si ottiene una payoff sbilanciata, sempre.

	A	C
A	2,1	0,0
C	0,0	1,2

Soluzione ipotetica: Sarebbe quello di avere un 50% di scelta di avere Comedy e 50% Action **randomizzata**.

13.1.4 Matching Pennies

Sarebbe **testa o croce**. (H o T) Come si può vedere dalla tabella **non c'è un equilibrio puro**. Il problema di questi giochi è che, in alcuni, **non esiste** un equilibro puro.

	H	T
H	1,-1	-1,1
T	-1,1	1,-1

La migliore strategia: Letteralmente conviene randomizzare la scelta tra **testa** o **croce**, per massimizzare la payoff.

13.2 Strategie e strategie miste

Definizione 13.3 *(Strategia)*

Una strategia s_i per un giocatore i è una **distribuzione di probabilità** su A_i.

Strategia Pura: Una sola azione viene giocata con probabilità 1.

Strategia Mista: Più di un'azione viene giocata con probabilità > 0.(Cioè non si ha una sola scelta che si fa per ogni mossa in ogni gioco). Le azioni con probabilità > 0 vengono chiamate **supporto** della strategia. **Nota:** con mosse con probabilità maggiore di 0 si indicano le possibili mosse da giocare. Indichiamo S_i l'insieme delle strategie del giocatore i.

Indichiamo con $S = S_1 \times S_2 \times \cdots \times S_n$ l'insieme delle strategie di tutti i giocatori.

Domanda 13.1 *(Come vengono calcolate le utilità nelle strategie miste?)*

Siccome le mosse non sono stabilite, cioè non è solo una mossa, **non possiamo** usare la **payoff matrix**, perché stiamo introducendo le **probabilità**. Ci vuole un altro modo per calcolarlo. Si utilizza l'**utilità attesa**.

$$\begin{aligned} u_{i(s)} &= \sum_{\alpha \in A} u_{i(\alpha)} Pr(\alpha|s) \\ Pr(\alpha|s) &= \prod_{j \in N} s_j(\alpha_j) \end{aligned} \tag{12}$$

Esempio 13.1 *(Testa o Croce)*

	H	T
H	1,-1	-1,1
T	-1,1	1,-1

Considernado che ogni posizione della matrice ha una possibilità 0.5.

$$u_r([\frac{1}{2}, \frac{1}{2}], [\frac{1}{2}, \frac{1}{2}]) = 1 \times 0.25 - 1 \times 0.25 - 1 \times 0.25 - 1 \times 0.25 = 0$$

Definizione 13.4 *(Migliore risposta con strategie miste) Supponiamo che il giocatore i conosca cosa giocherà l'altro giocatore*

$$s_i =< s_1, s_2, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n >$$

*La **migliore risposta con strategia mista** di s_i è la strategia che massimizza il payoff del giocatore i.*

$$s_i^* \in BR(s_i) \iff \forall s_i \in A_i, u_i(s_i^*, s_{-i}) \geq u_i(s_i, s_{-i})$$

Definizione 13.5 *(Equilibro di Nash Misto) Un profilo strategico si definsice **Equilibro di Nash Misto** se ogni giocatore gioca la migliore risposta:*

$$s =< s_1, s_2, \ldots, s_n >= NE \iff \forall i s_i \in BR(s_{-i})$$

Teorema: *Ogni gioco finito ha un Nash Equilibro.*

13.3 Nash Equilibria

Condizione per migliore risposta: Siano A e B le matrici di payoff for la riga e la colonna per players, rispettivamente. Una strategia s_r^* per la **giocatore riga** è la migliore risposta alla **strategia della giocatore colonna** s_c se e solo se:

$$s_{r,i}^* > 0 \implies (As_c^T)_i = \max(As_c^T) \forall i \in A_1$$

Nota: A_1 è l'insieme di azioni del giocatore riga e A_2 è l'insieme di azioni del giocatore colonna.

Praticamente, ci dice che la **strategia migliore per il giocatore riga** è quella che **massimizza** il **payoff** per ogni strategia del giocatore**colonna**.

Esempio 13.2 *Sassa, carta o forbice*

$$A = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$$

$$s_c = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ 0 \end{bmatrix}$$

$$As_c^T = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ 0 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}$$

Nota: As_c^T è il prodotto tra la matrice di payoff del giocatore riga e la strategia del giocatore colonna.

Esempio 13.3 *(Battle of Sexes)*

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \quad s_c = \begin{bmatrix} \frac{1}{2} \\ \frac{2}{2} \end{bmatrix}$$

Utilità giocatore colonna: $As_c^T = \begin{bmatrix} \frac{2}{3} & \frac{2}{3} \end{bmatrix}$

Migliore risposta:

$$s_r^* = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Esempio 13.4 *(Esempio con matrice)*

	C_1	C_2
R_1	**2**,**1**	**0**,**2**
R_2	1,1	1,3

Spiegazione: Praticamente, se il giocatore usasse la strategia di R_1, sceglierebbe quella **migliore**; in entrambi i casi, se il giocatore colonna usasse la colonna C_1 o C_2, il giocatore **riga** avrebbe un payoff maggiore rispetto ad usare la strategia di R_2. Quindi, la strategia migliore per il giocatore riga è quella di R_1.

13.3.1 Giochi a Somma-Zero

Siamo nel caso di giochi con 2 giocatori.

Definizione 13.6 *(Giochi a Somma-Zero)*

Un gioco con due giocatori (A,B) è definito a ***somma-zero*** *se* $\boldsymbol{A = -B}$.

Data una strategia x dal giocatore riga, il giocatore colonna può scegliere una strategia y che limita il payoff del giocatore riga.

Inversamente, data una strategia y del giocatore colonna, il giocatore riga punta a massimizzare la propria payoff.

Formalizzazione sotto forma di **problema lineare**:

Giocatore colonna:

$$\begin{aligned} \min_{v,y} \; & v \\ s.t. \; & \\ Ay^T & \leq \vec{1}v \\ y & \in S_2 \end{aligned} \tag{13}$$

- v: è il valore che il giocatore colonna vuole minimizzare
- y: è la strategia del giocatore colonna
- Ay^T: è la matrice di payoff del giocatore colonna
- S_2: è l'insieme delle strategie del giocatore colonna

Giocatore riga:

$$\begin{aligned} \max_{u,x} \; & u \\ s.t. \; & \\ Ax^T & \geq \vec{1}v \\ x & \in S_1 \end{aligned} \tag{14}$$

- u: è il valore che il giocatore riga vuole massimizzare
- x: è la strategia del giocatore riga
- Ax^T: è la matrice di payoff del giocatore riga
- S_1: è l'insieme delle strategie del giocatore riga

Esempio 13.5 *(Esempio con Sasso Carta o Forbice)*

$$\begin{aligned} \max_{u,x} \; & u \\ s.t. \; & \\ xA & \geq \vec{1}u \\ x & \in S_1 \end{aligned} \tag{15}$$

$$A = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$$

$$\begin{aligned} \max_{u,x} \; & u \\ 0x_1 + 1x_2 - 1x_3 & \geq u \\ -1x_1 + 0x_2 + 1x_3 & \geq u \\ 1x_1 - 1x_2 + 0x_3 & \geq u \\ x_1 + x_2 + x_3 & = 1 \end{aligned} \tag{16}$$

Portare in forma normale

$$\begin{aligned} &\min_x cx \\ &s.t. \\ M_{ub}x &\leq b_{ub} \\ M_{eq}x &= b_{eq} \\ x &\geq 0 \end{aligned} \tag{17}$$

$$\begin{aligned} &\min_x cx \\ 0x_1 - 1x_2 + 1x_3 1x_4 &\leq 0 \\ 1x_1 - 0x_2 - 1x_3 + 1x_4 &\leq 0 \\ -1x_1 + 1x_2 + 0x_3 + 1x_4 &\leq 0 \\ 1x_1 + 1x_2 + 1x_3 + 0x_4 &= 1 \end{aligned} \tag{18}$$

$$M_{ub} = \begin{bmatrix} 0 & -1 & 1 & 1 \\ 1 & 0 & -1 & 1 \\ -1 & 1 & 0 & 1 \end{bmatrix}, b_{ub} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$M_{eq} = \begin{bmatrix} 1 & 1 & 1 & 0 \end{bmatrix}, b_{eq} = 1, c = (0, 0, 0, -1)$$

Esempio 13.6 *(Equilibrio Misto con 2 giocatori)*

	A	C
A	2,1	0,0
C	0,0	1,2

Il giocatore 1 gioca A con probabilità p e C con probabilità $1 - p$
Il giocatore 2 cerca di rispondere al meglio al giocatore 1.
Giocatore 2 rende il giocarore 1 indifferente tra A e C

$$\begin{aligned} u_1(A) &= u_1(C) \\ 2p + 0(1-p) &= 0p + 1(1-p) \\ p &= \frac{1}{3} \end{aligned} \tag{19}$$

Il giocatore 1 gioca A con probabilità q e C con probabilità $1 - q$
Il giocatore 2 cerca di rispondere al meglio al giocatore 1.

Giocatore 1 rende il giocatore 2 indifferente tra A e C

$$\begin{aligned} u_2(A) &= u_2(C) \\ 1q + 0(1-q) &= 0q + 2(1p) \\ q &= \frac{2}{3} \end{aligned} \tag{20}$$

Le strategie miste $\left(\frac{1}{3}, \frac{2}{3}\right)$ e $\left(\frac{2}{3}, \frac{1}{3}\right)$ sono un **Equilibro di Nash**

14 Tree Decompositions Lab

Nozione: I problemi NP-Hard sugli **alberi** sono più semplici da risolvere rispetto ai grafi. Ma per quale motivo questa cosa funziona?

14.1 Maximum Independent Set (MIS)

Definizione 14.1 *(Independent Set)*
*Sia $G = (V, E)$ un grafo; Un **independent set** è un insieme di nodi tali che non ci sono archi tra loro.*

$$S = \{C \subset V, (u, v) \notin E \forall u, v \in C\}$$

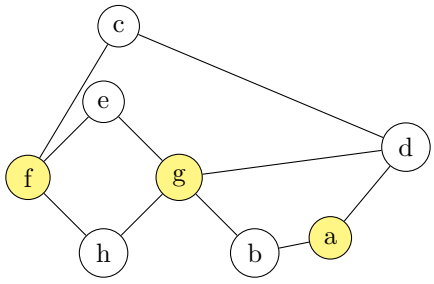


Figure 24: Independent Set formato da $\{f, g, a\}$

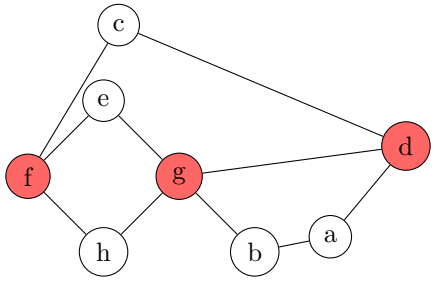


Figure 25: Independent Set Sbagliato

$$MIS = \arg\max_{s} |S|, \forall S \in IS(G)$$

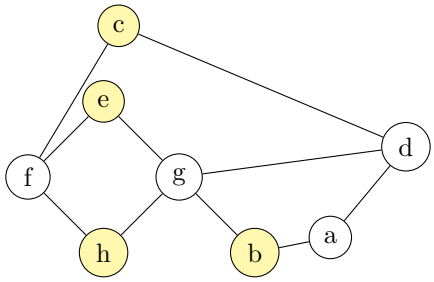
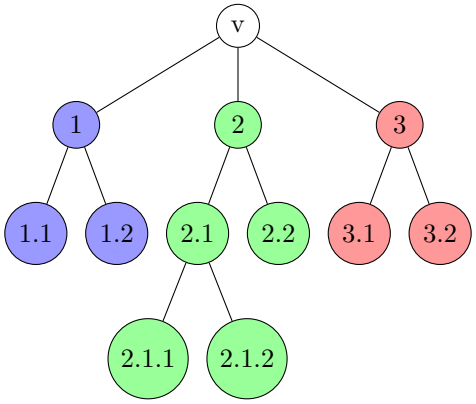


Figure 26: Maximum Independent Set formato da $\{e, h, c, b\}$

La soluzione per questo problema sfrutta la **programmazione dinamica** come approccio. Si risolvono dei sotto-problemi per poi risolvere il problema principale.

In generale, MIS è $NP - Completo$ e una implementazione Naive richiede $\mathcal{O}(n^2 2^n)$ come complessità temporale. Ma con gli **alberi** il problema si può risolvere in tempo **lienare**!



$$MIS(v) = MIS(T_{v,1}) + MIS(T_{v,2}) + MIS(T_{v,3}) \pm v$$

14.1.1 L'algoritmo di programmazione dinamica

Per ogni vertice v calcoliamo:

- $M^+[v] = |MIS(T_v) \cup \{v\}|$
- $M^-[v] = |MIS(T_v) \setminus \{v\}|$

Per un vertice v con figli $w_1, \ldots, w_k d$:

- $M^+[v] = 1 + \sum_{i=1}^{d} M^-[w_i]$
- $M^-[v] = \sum_{i=1}^{d} \max\{M^+[w_i], M^-[w_i]\}$

Quindi:

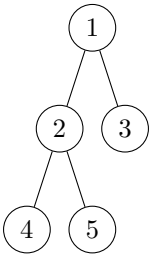
$$MIS(T) = \max\{M^+[r], M^-[r]\}$$

dove con **T** si intende l'albero completo.

Praticamente, quello che si fa è **calcolare il MIS per ogni sotto-albero** e poi si sommano i risultati per ottenere il **MIS dell'albero completo**.

L'algoritmo, praticamente, parte dalle **foglie** e risale verso la **radice**.

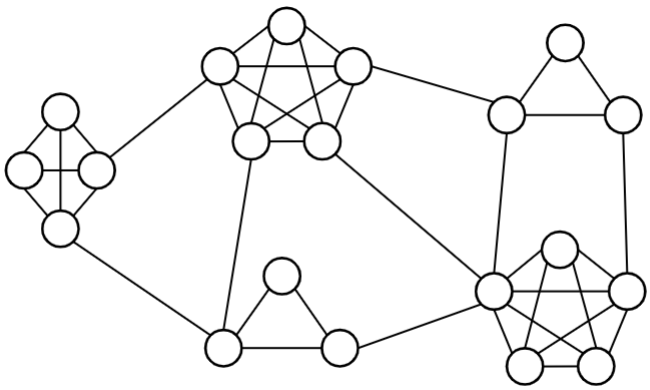
Esempio 14.1 *(MIS su un albero facile)*



Eseguendo l'algoritmo si ottiene:

- **Nodo 4**:
  - $M^+[4] = 1$
  - $M^-[4] = 0$
- **Nodo 5**:
  - $M^+[5] = 1$
  - $M^-[5] = 0$
- **Nodo 2**:
  - $M^+[2] = 1$
  - $M^-[2] = 2$
- **Nodo 3**:
  - $M^+[3] = 1$
  - $M^-[3] = 0$
- **Nodo 1**:
  - $M^+[1] = 1 + 2 = 3$
  - $M^-[1] = 3$

E cosa ci muoviamo in un caso del genere?



Se notiamo, possiamo dividere il problema in sotto-problemi.

Intuizione: Possiamo rappresentare un **grafo** G come un albero T. I nodi di T sono dei piccoli moduli che chiamiamo **bags**. In questo caso, li trattiamo come *sotto-problemi*.

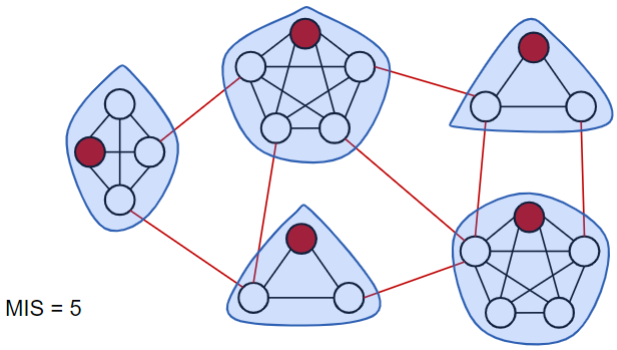
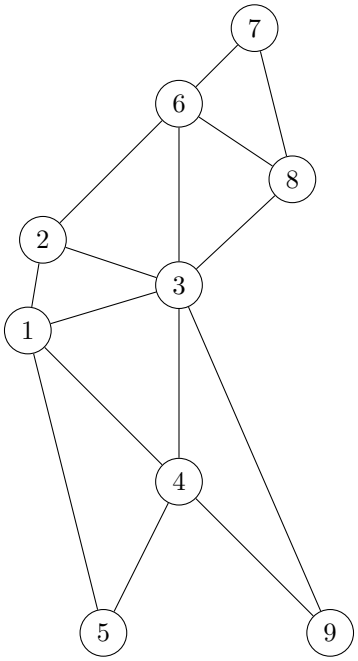
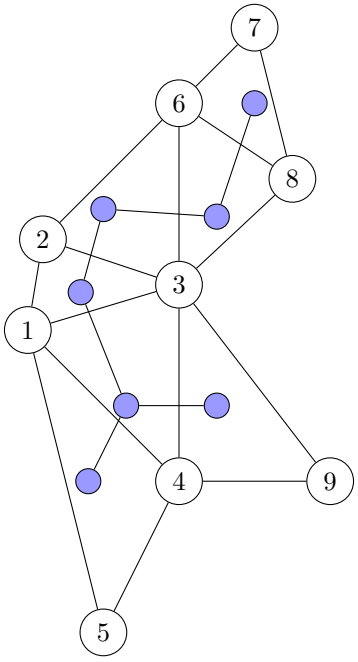


Figure 27: Formato del problema

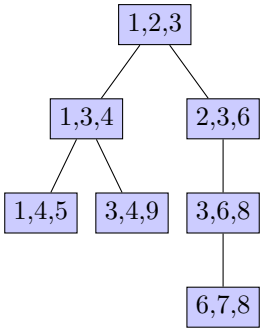
Utilizziamo la programmazione dinamica per risolvere la tree decomposition.
Copiare grafico di Tree Decomposition: intuition



L'intuizione: Si prendono praticamente gli insiemi di nodi che formano una **cricca** tra loro. In questo caso, nel nostro grafo:



E praticamente si va a formare una **tree decomposition** di questo tipo:



Definizione 14.2 *(Tree Decomposition) Una tree decomposition di un grafo* $G = (V, E)$ *è un albero* T *di bags* X*:*

- *se* $(u, v) \in E$ *allora* u *e* v *sono nella stessa bag.*
- $\forall v \in V$ *i bags che contengono* v *sono connessi in* T

Un grafo può ammettere diverse tree decomposition.

Definizione 14.3 *(Tree width) La treewidth è la width minore possibile tra tutti le tree decomposition ammesse nel grafo.*

Un po' di notazione:

- $tw(G)$ = *treewidth di* G
- $tw(G) = 1 \iff G$ *è una* ***foresta***
- $tw(G) = 2 \iff G$ *è una serie di grafi paralleli*
- *Eliminare un arco in* G *non aumneta la treewidth*
- *Contrarre archi non aumenta la treewidth*
- *Ogni cricca in* G *deve essere contenuta in una bag*

14.2 Cops e Robber

Abbiamo questo problema che viene definito in questo modo.

- **1 Ladro** che si muove sul grafo.
- k poliziotti che si muovono sul grafo.
- Per vincere, i poliziotti devono catturare il ladro e arrivare al suo nodo.

Teorema 14.1 *(Cops e Robber Condition)* $tw(g) \leq k \iff k+1$ *poliziotti possono vincere il gioco.*
La strategia è data dalla ***tree decomposition***

Usando il grafo di prima, e successivamente anche la tree decomposition che è stata calcolata prima, possiamo vedere come effettivamente viene risolto il problema di cops e robber.

Esempio 14.2 *(Cops e Robber)*

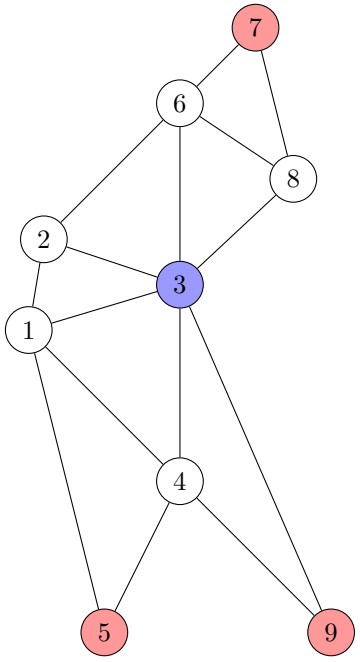


Figure 28: It 1 Grafo

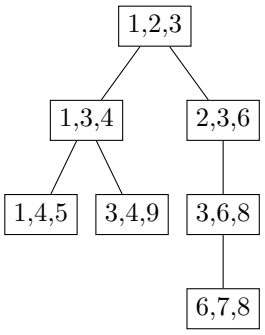


Figure 29: It 1 Tree Decomposition

In questo caso, la strategia è spostare i ***poliziotti*** *che sono i nodi* ***rossi*** *seguendo la tree decomposition, arrivando a catturare il* ***ladro*** *che è il nodo* ***blu****.*

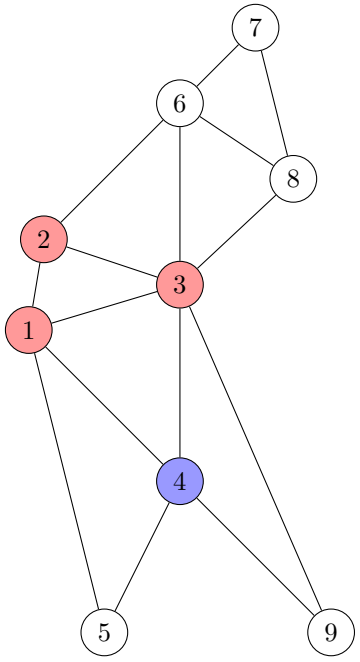


Figure 30: It 2 Grafo

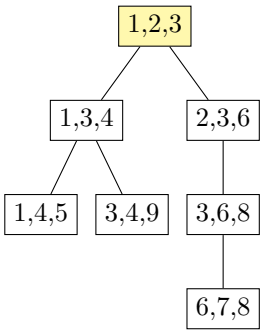


Figure 31: It 2 Tree Decomposition

Si continua ad applicare, ogni volta, la strategia in base alla tree decomposition.

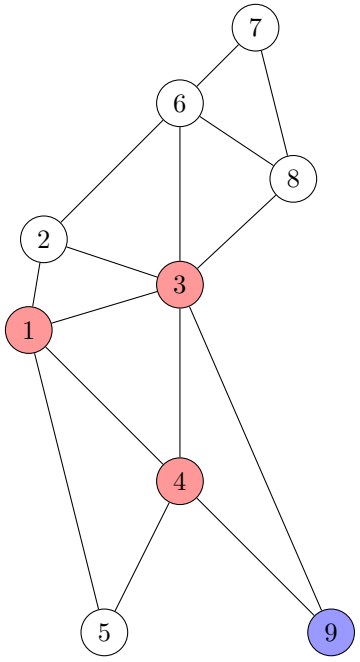


Figure 32: It 3 Grafo

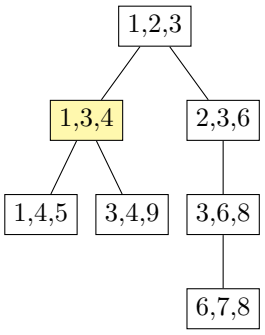


Figure 33: It 3 Tree Decomposition

Ora, seguendo l'ultimo passaggio, si arriva alla fine del gioco e i poliziotti hanno vinto.

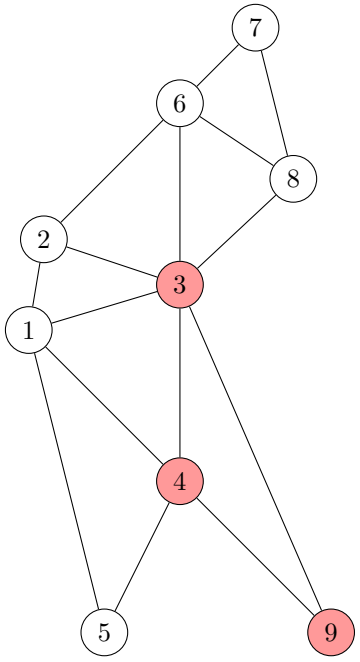


Figure 34: It Final Grafo

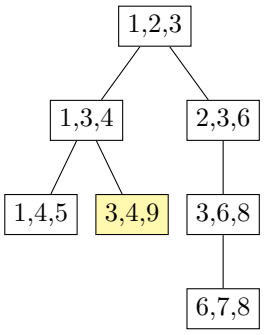


Figure 35: It Final Tree Decomposition

14.3 Calcolare la tree decomposition

Si usano 2 concetti principali: **rimuovere un nodo** e **triangolazione dei vicini** per costruire le bags.

Definizione 14.4 *(Triangolazione dei vicini e rimozione dei nodi) Un grafo G è **triangolato** se il ciclo più piccolo all'interno del grafo ha lunghezza 3. (Da controllare)*

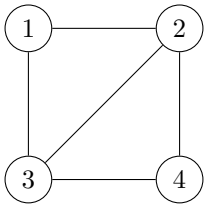


Figure 36: Triangolazione

Per poter rimuovere un nodo v, se nel grafo G rimuoviamo questo v, tutti i vicini di v $neig(v)$ devono essere tutti connessi tra loro.

Ad esempio, nel grafo sopra, possiamo rimuovere in un caso il nodo 1 *e nel secondo caso il nodo* 4*.*

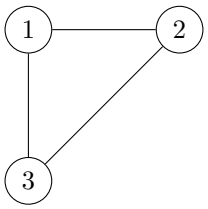


Figure 37: Triangolazione senza Nodo 4

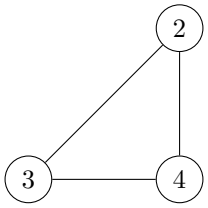


Figure 38: Triangolazione senza Nodo 1

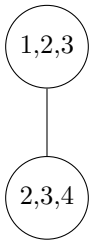


Figure 39: Perfect Elimination Order

I nodi che possono essere rimossi si chiamano ***simplicial node****.*
Quindi, la migliore tree decomposition con la migliore treewidth si riduce a trovare la ***PEO*** *(Perfect Elimination Order)*

Torniamo ora al **calcolo** della tree decomposition. Partiamo da un **removal order** per un grafo.
Disegnare il grafo o mettere qualcos

$$removal\_order = [6, 7, 9, 1, 8, 2, 0, 3, 5, 4]$$

Praticamente, $\forall v \in removal\_order$: rimuovi v e triangola i vicini. Se i vicini sono già triangolati, si *crea* la *bag* che avrà v e i suoi vicini come nodi.

Si parte questo grafo in foto e mano a mano si cominciano a togliere nodi in base a quell'ordine indicato.

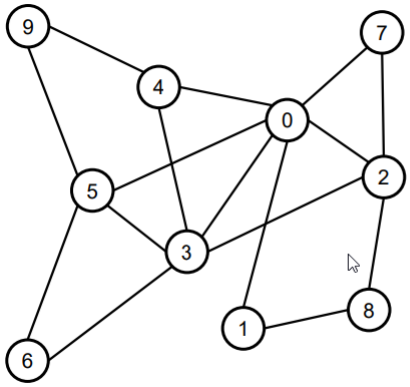


Figure 40: Grafo iniziale

Ad esempio, se si partisse dal nodo 6 si avrebbe **rimuovendolo** i suoi vicini 5 e 3 sarebbero già triangolati, ovvero connessi tra loro. Quindi, siccome non serve fare altro, si crea la bag $\{6, 5, 3\}$

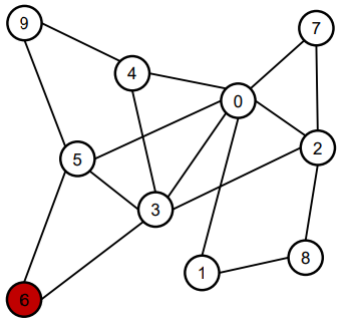


Figure 41: Esempio rimozione nodo 6

$$\forall v \in removal\_order : \begin{cases} \text{Rimuovi il nodo} \\ \text{I vicini sono già triangolati?} \\ \begin{cases} \text{Si: non fare niente} \\ \text{Altrimenti: triangola i vicini} \end{cases} \end{cases} \tag{21}$$

Nota: Quando si arriva a $\{3, 4, 5\}$ notiamo che esise già una bag più grande che è $\{0, 3, 4, 5\}$ e non ci serve quindi crearne un'altra. Dopo aver creato tutte le bag, **si connette ogni bag** con la quale l'intersezione è massimale. Cioè, ogni bag è connessa a quell'altra bag tale che il numero di nodi in comune è massimo.

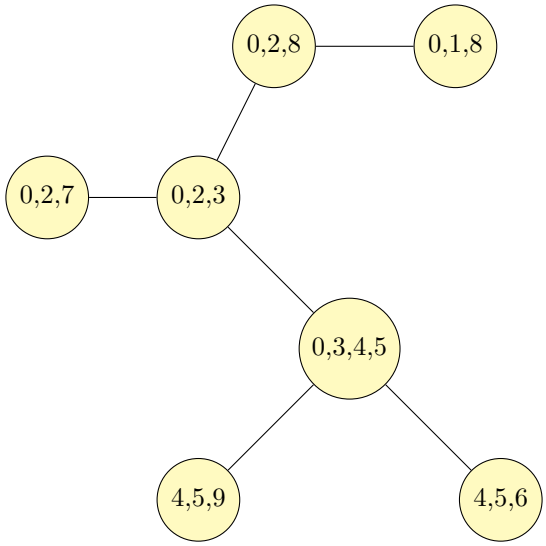


Figure 42: Tree Decomposition dal Removal Order

Risultato: Abbiamo ottenuto una tree decomposition!

La domanda ora è: **Come otteniamo il *removal order*?**

Il **calcolo** del removal order è un problema **NP-Hard**. Ci sono $N!$ possibili ordini e $\mathcal{O}(2^n)$ con la programmazione dinamica.

Per risolvere questo problema, quindi, è usare un'euristica per trovare delle **soluzioni accettabili**.

Definizione 14.5 *(Minimum degree heuristic) Quando rimuoviamo un nodo dalla tree decomposition, rimuoviamo il nodo che ha il* ***grado minore****, cioé il nodo che ha il* ***minor numero di vicini****.*

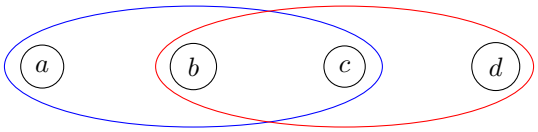
Rimuovere un nodo cambia anche il ***grado*** *dei vicini e quindi calcoliamo il* ***minimum degree*** *ad ogni iterazione.*

Di fatto, il removal order di prima è stato calcolato sfruttando questa **minimum degree heuristic**.

15 Tree Decomposition e programmazione dinamica

Definizione 15.1 (Hypergraph) *Un* hypergraph *è una coppia* $H = (V, E)$ *dove* V *è un insieme di* vertici *e* $E \subseteq 2^V$ *è un insieme di* iper-archi.

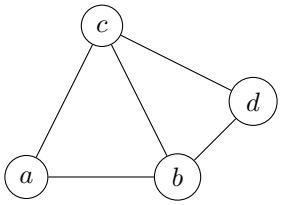
Un hyperedge *è un arco che collega più di due vertici.*



Indichiamo con h *un iper-arco.*

- $h_1 = a, b, c$
- $h_2 = b, c, d$

Possiamo indicare questo con un grafo primale.



15.1 MIS sui grafi con treewidth limitata

Idea: Per ogni coppia di bag **genitore-figlio**, dobbiamo gestire la loro intersezione.

Ad esempio, date due base $B_1 = \{B, C, D\}$ e $B_2 = \{C, D, E\}$, la loro intersezione è $B_1 \cap B_2 = \{C, D\}$. Bisogna risolvere il *sottoproblema* per ogni *sottoinsieme*: $\{D\}, \{C\}, \{E\}, \{C, D\}, \{C, E\}, \{D, E\}, \{C, D, E\}$.

Diamo qualche definizione di annotazione:

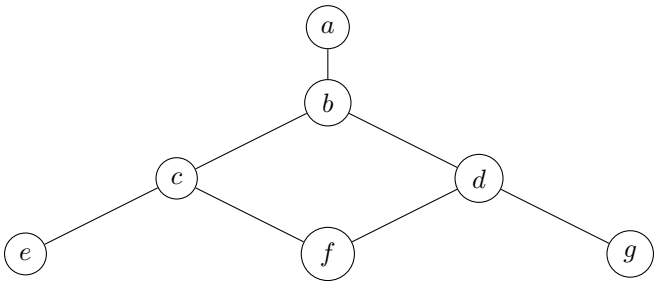
$$\begin{aligned} \text{La radice} &= R \\ \text{Una bag} &= X_i \\ \text{Figlio di } X_i &= X_j \\ \text{Genitore di } X_i &= X_{k,i} \end{aligned} \tag{22}$$

$$A[S,i] = MIS(S) \forall S \in X_i + \sum_j MIS(X_j)$$

$$B[S,i] = \max_{S' \subset X_i} A[S',i], \quad S = S' \cap X_{k,i}$$

$$MIS(G) = \max_{S \subset R} B[S,R]$$

Esempio 15.1 *(Esempio con grafo di prima)*



Riscriviamo le formule cosi:

$$\begin{aligned} A[S,i] &= |S| + \sum_j B[S \cap X_{k,i}, j] - |S \cap X_J| \\ B[S,i] &= \max_{S' \subset X_i} A[S',i] \quad S = S' \cap X_{k,i} \end{aligned} \tag{23}$$

Ora, cerchiamo capire come funziona il tutto.
$B[\mathcal{Z}, i]$.
$\max_{S \subset X_i} Z = S \cap X_{k,i}$.

- $\emptyset = 0 \rightarrow$ Z $= \emptyset$
- C $= 1$
- E $= 1 \rightarrow = \emptyset$
- F $= 1$
- CF = Sono connessi, quindi non calcolo l'IS $-\infty$
- CE = Sono connessi, quindi non calcolo l'IS $-\infty$
- CEF = Sono connessi, quindi non calcolo l'IS $-\infty$

Quindi, praticamente, vediamo l'interseizone diogni sotto insieme con il bag padre, e vediamo per quella funzione Z = intersezione bag padre e sottoinsieme contenenti i precedenti indipendent set, prendiamo il valore di A massimo con la Z che coincide.

Tutto questo SENZA CONSIDERARE LA SOMMATORIA I A perché eravamo nei nodi foglia. Da ampliare dopo. Commento e poi modifico.

Partiamo dal basso con C E F. Si calcola A di A[empty] = 0, A[C] = A[E] = A[F] = 1. Poi B[EMPTY] = 1, B[C] = B[F] = 1.

Poi passa a D F G . Uguale. A[EMPTY] = 0, A[D] = A[F] = A[G] = 1. Poi B[EMPTY] = 1, B[D] = B[F] = B[D] = 1.

Ora, andiamo a C D F. A[EMPTY]. La cardinalità è chiaramento 0. Ora c'è la sommatoria $\sum_j B[S \cap X_{k,i}, j] - |S \cap X_J|$. La prima parte è chiaramente B[$\emptyset$] e poi c'è la sottrazione con la cardinalità di $\emptyset$ che è 0. Alla fine avremo $0 + 1 - 0 + 1 - 0$, che è 2.

Vediamo ora $A[C]$. La cardinalità è 1. Ora c'è la sommatoria $\sum_j B[S \cap X_{k,i}, j] - |S \cap X_J|$. La prima parte $B[A \cap CEF]$ e la seconda $|S \cap CEF|$. Ora, c'è un altro figlio e quindi la sommatoria continua con $B[A \cap DFG] - |S \cap DFG|$. Qui l'intersezione è vuota quindi $B[\emptyset] = 1$ e $|\emptyset| = 0$.

La somma finale è: $1 + 1 - 1 + 1 - 0 = 2$.

E si continua ad oltranza.