Alma Mater Studiorum · Università di Bologna

DIPARTIMENTO DI MATEMATICA
Corso di Laurea in Matematica

LA GERARCHIA POLINOMIALE: TRA PROBLEMI APERTI E TEORIA DEI GIOCHI COOPERATIVI

Tesi di Laurea in Informatica Teorica

Relatore:
Chiar.mo Prof.
MALIZIA ENRICO

Presentata da:
GIORDANI FABIO

Anno Accademico 2024-2025

A tutte le Mamme del mondo...

Introduzione

Da quando fu costruito il primo computer, l'uomo ha cominciato ad apprezzarne ed esplorarne le potenzialità. In questo contesto risultò naturale la nascita della disciplina della complessità computazionale, la quale si occupa di studiare quante risorse in termini di spazio e di tempo i vari problemi dell'informatica necessitino per la loro soluzione. Fu così che ebbero origine le prime ***classi di complessità*** che riunivano i problemi in base alla loro complessità (temporale e spaziale). La complessità strutturale nacque poi con l'obiettivo di analizzare le relazioni e i contenimenti tra le varie classi di complessità.

In questo breve elaborato presenteremo una famiglia di classi di complessità temporale, la ***gerarchia polinomiale***, vedremo alcune sue curiose ed interessanti proprietà ed esamineremo alcuni problemi nati dalla teoria dei giochi cooperativi e collocabili proprio in tale famiglia. Il primo capitolo sarà una rapida introduzione alla teoria della complessità computazionale, mentre nei successivi entreremo nel cuore dei vari argomenti: il secondo sarà incentrato appunto sulla gerarchia polinomiale (si faccia particolare attenzione alle sue caratteristiche peculiari, che la legano a doppio filo al famoso problema del millennio ***P vs NP***); il terzo ed ultimo capitolo tratterà di teoria dei giochi cooperativi e in esso ci concentreremo sui concetti di Bargaining Set[1] e di Kernel[1] di un gioco a grafo e vedremo come la computazione di tali concetti di soluzione sia classificabile proprio nella gerarchia polinomiale.

Per approfondimenti e chiarimenti si tenga sempre presente la bibliografia al termine.

[1]Ho scelto di usare la lingua italiana per la maggior parte dei termini dell'elaborato, ad eccezione di questi due, per la loro rilevanza nella letteratura, e pochi altri. Gli acronimi sono invece in gran parte calcati dall'inglese.

Indice

Capitolo 1

Nozioni di base

In questo primo capitolo introduciamo brevemente gli strumenti di base per studiare la complessità strutturale, vale a dire le Macchine di Turing (T.M.) e la teoria dei linguaggi.

1.1 Stringhe, problemi, linguaggi

Definizione 1.1.1.

1. Un **ALFABETO** è un qualunque insieme finito e non vuoto. Di solito denoteremo un alfabeto con la lettera greca maiuscola Σ.
2. L'alfabeto $\Sigma = \{0, 1\}$ viene detto **BINARIO**.
3. Un **SIMBOLO** (o lettera) è un elemento di un alfabeto.
4. Dato un alfabeto Σ, una **STRINGA** (o parola) su Σ è una sequenza finita di simboli di Σ. Denoteremo una stringa con la lettera w generalmente.
5. La stringa vuota ε è l'unica parola che consiste di zero lettere di Σ.
6. Data una stringa w, la sua **LUNGHEZZA** (o taglia) è il numero di elementi di cui è composta. La denotiamo $|w|$.
7. Date due parole v e w su Σ, si definisce la **CONCATENAZIONE** di v e w, denotata con $v \cdot w$, come la parola z che consiste dei simboli di v nello stesso ordine seguita dai simboli di w nello stesso ordine.

Definizione 1.1.2. Data una parola w su Σ e un naturale n, definiamo w^n induttivamente come: $\begin{cases} w^0 = \varepsilon \\ w^{n+1} = w \cdot w^n \quad \text{per ogni } n \geq 0. \end{cases}$

Definizione 1.1.3. L'insieme di tutte le parole su un alfabeto Σ verrà denotato Σ^*.

Definizione 1.1.4. Dato un alfabeto Σ, un **LINGUAGGIO** su Σ è un sottoinsieme di Σ^*. Chiamiamo **CLASSE** qualunque collezione di linguaggi.

Definizione 1.1.5. Dato un linguaggio L di alfabeto Σ, il **COMPLEMENTO** di L è denotato $\bar{L}$ ed è definito con la seguente differenza insiemistica:

$$\bar{L} \coloneqq \Sigma^* \setminus L$$

Più in generale sono definite in modo usuale le operazioni insiemistiche tra classi e linguaggi (unione, intersezione...).

La locuzione "***Decidere un linguaggio***" significa mettere a punto un algoritmo che riceve in input stringhe e risponde SI o NO in tempo finito con il seguente criterio: viene risposto SI se e solo se la stringa di input appartiene al linguaggio e viene risposto NO in caso contrario.

Definizione 1.1.6. Un **PROBLEMA DI DECISIONE** in informatica è una funzione che associa ad ogni stringa di un alfabeto Σ uno e un solo valore tra 0 e 1. Un **PROBLEMA DI LINGUAGGIO** è un problema di decisione in cui l'obiettivo è stabilire se ogni stringa w di Σ^* appartiene a un certo linguaggio L o meno (con $L \subseteq \Sigma^*$).

Lemma 1. Ogni problema di decisione può essere rappresentato con un problema di linguaggio.

Dimostrazione. Semplice verifica: sia $f : \Sigma^* \longrightarrow \{0, 1\}$ un problema di decisione. Poniamo $L_f = \{w \in \Sigma^* \mid f(w) = 1\}$. Chiaramente risolvere il problema f equivale a rispondere alla domanda $w \in L_f$. □

Questo permette di studiare pù uniformemente tutti i problemi di decisione trattandoli come problemi di linguaggio, il che ha grossi vantaggi, soprattutto in termini di formalizzazione e modellizzazione di algoritmi per risolverli, come vedremo nel prossimo paragrafo. In particolare useremo come sinonimi le parole "problema" e "linguaggio".

1.2 Macchine di Turing deterministiche e trasduttori

Il formalismo più usato per modellizzare algoritmi è quello delle Macchine di Turing, che ora presentiamo per completezza e uniformità di notazione. Intuitivamente una macchina di Turing (TM) è un apparecchio che consiste di:

1. un insieme finito di stati interni;
2. uno o più **NASTRI DI LAVORO** semi-infiniti, suddivisi in **CELLE**, ognuno dei quali munito di una **TESTINA** che si può muovere a destra o a sinistra, leggere le celle del nastro - una per volta - ed anche sovrascrivere simboli;
3. un ulteriore nastro speciale con corrispondente testina, detto **NASTRO DI INPUT**, che invece è di sola lettura.

Il termine "semi-infinito" per indicare un nastro significa che quest'ultimo non ha una cella più a destra (in tale direzione non termina), ma ha una cella più a sinistra: se la testina si trova su di essa, non può muoversi ulteriormente a sinistra.

Una macchina M comincia il suo calcolo su una stringa w con il nastro di input contenente w (un simbolo in ogni cella, in ordine, a partire dalla cella più a sinistra) e con un simbolo speciale detto "Blank" (b) in tutte le celle rimanenti. Anche tutte le celle dei nastri di lavoro contengono b all'inizio. In ogni istante la macchina si trova in uno degli stati. La macchina è in grado di leggere i simboli delle celle su cui si trova la testina di ciascun nastro di lavoro, riscrivere su tali celle nuovi simboli (sovrascrivendo i precedenti), muovere ciascuna testina a destra o a sinistra e cambiare il proprio stato interno. Tutte queste operazioni vengono effettuate in blocco, formano uno **STEP** e sono univocamente determinate dalla **FUNZIONE DI TRANSIZIONE** della macchina, la quale dipende dallo stato interno della macchina e dai simboli letti sui nastri. Si noti che il nastro di input può essere letto ma non sovrascritto.

Appena una macchina riceve un input, si assume che il suo stato interno sia lo stato iniziale q_0. Dopodiché essa procede applicando la funzione di transizione, step dopo step, finché possibile: ogni qualvolta la funzione di transizione non è definita, la macchina si ferma. Se alla fine degli step la macchina si trova in uno stato accettante, diremo che M accetta la stringa w, altrimenti M rifiuta w. Si tenga presente che esistono due modi per rifiutare: o la macchina si ferma in uno stato non accettante o non si ferma affatto, continuando a fare infiniti step uno dopo l'altro.

Vediamo una definizione più formale:

Definizione 1.2.1. Una **MACCHINA DI TURING DETERMINISTICA** M a k nastri è una 5-upla

$$M = (Q, q_0, F, \Sigma, \delta) \text{ dove:}$$

1. Q è un insieme finito degli stati interni;
2. $q_0 \in Q$ è lo stato iniziale;
3. $F \subseteq Q$ è insieme degli stati accettanti;
4. Σ è alfabeto di input e di nastro;
5. $\delta : Q \times \Sigma^k \longrightarrow Q \times \Sigma^{k-1} \times \{R, N, L\}^k$ è una funzione (spesso parziale): la funzione di transizione della macchina M. Situazioni di calcolo in cui la funzione di transizione non è definita indicano che la macchina si blocca in tal punto. In caso contrario, il risultato della funzione di transizione si interpreta come segue: la prima componente è il nuovo stato, la seconda sono i $k-1$ simboli da sovrascrivere ordinatamente nei nastri di lavoro e la terza componente indica il movimento che ciascuno dei k nastri esegue (R significa a destra di una cella, N significa nessun movimento e L significa a sinistra di una cella se possibile, altrimenti nessun movimento).

Definizione 1.2.2. Data una macchina di Turing M, una **CONFIGURAZIONE** per M è una descrizione del calcolo in un certo istante di tempo: essa include il contenuto dei nastri, la posizione di ciascuna testina e lo stato in cui la macchina si trova. Se M ha k nastri, una configurazione per M è una $k+1$-upla

$$(q, x_1, \ldots, x_k)$$

dove q è lo stato corrente di M e per ogni $j \in \{1, \ldots, k\}$, $x_j \in \Sigma^* \# \Sigma^*$ rappresenta il contenuto del j-esimo nastro. Il simbolo # (non lettera di Σ), indica la posizione della testina e per convenzione precede il simbolo che la testina legge in tale istante. Tutti i simboli dei nastri infiniti che non compaiono in x_j sono assunti b.

Definizione 1.2.3. La **CONFIGURAZIONE INIZIALE** di una macchina M sulla stringa w è $(q_0, \#w, \#, \cdots, \#)$.

Definizione 1.2.4. Una **CONFIGURAZIONE ACCETTANTE** è una configurazione il cui stato è accettante.

Definizione 1.2.5. Data una macchina di Turing M e una stringa di input w, una **COMPUTAZIONE PARZIALE** di M su w è una sequenza (finita o meno) di configurazioni di M in cui ogni step tra una configurazione e l'altra obbedisce alla funzione di transizione di M. Una **COMPUTAZIONE** di M su w è una computazione parziale che inizia con la configurazione iniziale di M su w e termina con una configurazione in cui nessuno step può essere eseguito. Una computazione viene detta **ACCETTANTE** se termina in una configurazione accettante: in tal caso l'input viene accettato da M.

Definizione 1.2.6. Il **LINGUAGGIO** di una macchina di Turing deterministica M è l'insieme delle parole accettate da M e viene denotato $\mathcal{L}(M)$. Spesso useremo le espressioni linguaggio "accettato" da una macchina di Turing e linguaggio "deciso" da una TM in modo equivalente[1].

Talvolta vorremmo usare TM per calcolare funzioni. Possiamo ottenere questo risultato aggiungendo alla macchina di Turing un **NASTRO DI OUTPUT**, con la condizione che esso sia di sola scrittura (vale a dire che la testina può solo scrivere simboli e muoversi verso destra o stare ferma, non "leggere" simboli o spostarsi a sinistra). Una tale macchina è detta **TRASDUTTORE**.

Dato un trasduttore M, la funzione f che esso calcola è definita su $\mathcal{L}(M)$ in questo modo:

$$f : \mathcal{L}(M) \longrightarrow \Sigma^*$$

per ogni $w \in \mathcal{L}(M)$, $f(w)$ è la parola che compare nel nastro di output quando M termina la computazione di w. I concetti di configurazione e computazione per i trasduttori sono quelli per macchine di Turing normali, con l'aggiunta del contenuto del nastro di output.

Esistono molte varianti di macchine di Turing, alcune delle quali le introduciamo nei prossimi paragrafi. Inoltre esistono svariati altri modelli di calcolo (λ-calcolo, funzioni ricorsive, sistemi combinatori di Post, algoritmi normali di Markov...) tutti equivalenti alla TM: una tale abbondanza suggerisce che le macchine di Turing corrispondano al più naturale e intuitivo concetto di "algoritmo". Di qui in avanti sarà dunque assunto il seguente fondamentale assioma:

Tesi di Church-Turing. Qualunque algoritmo può essere descritto da una macchina di Turing.

[1] Ai nostri fini tali parole sono usate come sinonimi: in letteratura, soprattutto in teoria della decidibilità, essi sono termini **distinti**.

Osservazione. Il modello di computazione dato dalla macchina di Turing non è facile da padroneggiare: presentare algoritmi sotto forma di TM è spesso arduo e comporta un grosso sacrificio in intellegibilità, al pari di lavorare con il codice di macchina di un computer. Per questa ragione ci appelleremo alla Tesi di Church-Turing e descriveremo di volta in volta il funzionamento di una TM non con complessi grafi orientati ed etichettati o artifici simili, ma con uno pseudocodice di alto livello molto più comprensibile al lettore, o addirittura con una semplice descrizione del funzionamento.

1.3 Macchine non deterministiche

Nella sezione 1.2, ogni step (o passo) di una macchina è completamente determinato dalla configurazione precedente: lo stato e i simboli letti dalle testine fissano lo step successivo. Ora alleggeriamo questa condizione ottenendo macchine non deterministiche, le cui "prossime mosse" possono essere decise tra alcune possibilità.

Definizione 1.3.1. Una macchina d Turing M **NON DETERMINISTICA** a k nastri è una 5-upla

$$M = (Q, q_0, F, \Sigma, \delta)$$

dove tutti gli elementi sono identici al caso deterministico tranne che la funzione di transizione, la quale ora è data da: $\delta : Q \times \Sigma^k \longrightarrow Q \times \Sigma^{k-1} \times \mathcal{P}\left(\{R, N, L\}^k\right)$, dove per ogni insieme A, $\mathcal{P}(A)$ indica il suo insieme delle parti.

Le definizioni ed osservazioni fatte per il caso deterministico rimangono inalterate, tuttavia ora su un certo input w non vi è un'unica computazione possibile, ma un insieme. Occorre dunque modificare la condizione di accettazione:

Definizione 1.3.2. Un input w viene accettato da una macchina non deterministica M se e solo se esiste una computazione di M su w che termina con uno stato accettante.

Osservazione. Si noti come non sia davvero utile permettere a una macchina non deterministica di avere un numero di possibili nuove configurazioni maggiore di due: infatti una scelta tra k possibilità equivale a $\lceil log\ k \rceil$ scelte tra due possibili vie. Pertanto senza perdere di generalità possiamo assumere che $|\delta(q, a)| \leq 2$ per ogni $(q, a) \in Q \times \Sigma^k$.

In questo modo l'insieme di tutte le possibili computazioni di una TM non deterministica su una stringa di input w può essere rappresentato da un albero

binario, detto **ALBERO DI COMPUTAZIONE**. La sua radice è la configurazione iniziale di M su w e induttivamente i figli di ciascun nodo c saranno le configurazioni che possono essere raggiunte da c in un passo applicando la funzione di transizione di M. Le foglie dell'albero saranno tutte le possibili configurazioni finali. Per definizione, un input è accettato se e solo se vi è almeno una foglia accettante nell'albero di computazione di tale input.
In altre parole una macchina non deterministica accetta "se ha un modo di accettare".

Osservazione. Un modo usuale per costruire una macchina non deterministica che svolga un certo calcolo è quello comunemente detto di ***"guess and check"***. Esso consiste nell'identificare l'elemento chiave (o gli elementi chiave) per risolvere il problema e fare in modo che il non determinismo della macchina lo faccia "comparire" sul nastro: dopodichè gli step successivi dovranno essere studiati per verificare che ciò che è stato indovinato sul nastro sia corretto. Se nessuno tra tutti i possibili "guess" è stato valutato come corretto, la macchina rifiuta perché non ha trovato un modo per accettare.
Ad esempio per stabilire se un certo numero abbia divisori non banali, basta considerare la macchina non deterministica M seguente:
ricevuta in input una stringa binaria,

- scrive sul nastro non deterministicamente dei caratteri binari (il non determinismo si evidenzia quando la macchina ha la duplice possibilità di scrivere oppure di passare allo stato successivo);
- controlla che ciò che è stato scritto sul nastro sia compreso tra 2 e l'input e divida la stringa in input (assumiamo che nella macchina sia presente una subroutine che lo faccia, che non approfondiamo);
- in caso di risposta affermativa, la macchina accetta la stringa poiché ha trovato un divisore non banale;
- in caso di risposta negativa, la macchina si ferma e rifiuta.

Vediamo ora che rapporto c'è tra le macchine deterministiche e quelle non deterministiche.

Lemma 2. È possibile codificare in linguaggio binario la funzione di transizione di una qualunque macchina di Turing. In particolare qualunque TM può essere indicata con un numero naturale (la stringa binaria della sua codifica) e dunque esiste solo una quantità al più numerabile di algoritmi.

D'ora in avanti parleremo senza specificare indifferentemente di TM e delle loro codifiche binarie: a quale delle due faremo riferimento sarà di volta in volta chiaro dal contesto. Questo discorso ha conseguenze importanti, che ora andiamo solo a richiamare:

1. è possibile passare in input una macchina di Turing a un'altra;
2. esiste una macchina **UNIVERSALE** che prende in input coppie (macchina M-stringa w) e risponde esattamente come la macchina M avrebbe risposto su w impiegando un numero di step quadratico. Si dice che la macchina universale è in grado di ***simulare*** altre macchine;
3. è possibile simulare una TM non deterministica con una deterministica performando un numero di passi esponenziale, esplorando l'albero di computazione livello per livello alla ricerca di configurazioni accettanti.

1.4 Macchine ad oracolo

Andiamo ora a definire macchine di Turing capaci di operare grazie a un aiuto esterno.

Una macchina a oracolo $M^?$ è una TM che ha la capacità speciale di poter interpellare un **ORACOLO** - altro non è che un linguaggio fissato - riguardo una stringa, ottenendo istantaneamente risposta SI o NO in base all'appartenenza o meno di tale stringa all'oracolo e traendo così informazioni utili per la computazione. La struttura e funzione di transizione della macchina è indipendente da quale linguaggio-oracolo si scelga (chiaramente il linguaggio globalmente accettato dalla macchina varierà). Quando si indica una macchina a oracolo senza aver specificato l'oracolo, detto anche oracle set, si usa la notazione $M^?$. Viceversa se l'oracle set è stato dichiarato come il linguaggio L si indica M^L. Più precisamente il concetto si può formalizzare come segue:

Definizione 1.4.1. Una **MACCHINA A ORACOLO** $M^?$ con oracle set L è una macchina di Turing (deterministica o meno) dotata di un nasto ulteriore, detto **NASTRO D'ORACOLO**, di sola scrittura, e di 3 stati specifici $(q_?, q_{yes}, q_{no})$ il cui funzionamento è il seguente:

1. M^L funziona come una normale macchina di Turing, purché non transisca nello stato $q_?$;
2. quando M^L transisce nello stato $q_?$, il contenuto (una certa stringa v) del nastro d'oracolo viene istantaneamente cancellato e M^L transisce nello stato q_{yes} se $v \in L$, q_{no} in caso contrario;

3. M^L non può transire verso gli stati q_{yes} e q_{no} in modo diverso da quello descritto al punto precedente.

Si noti come una qualunque TM normale è di fatto una macchina a oracolo con oracle set vuoto (equivalentemente, che non passa mai per lo stato $q_?$). Operativamente, possiamo pensare a macchine a oracolo come a macchine di Turing capaci di aiutare il proprio calcolo affidandosi una routine esterna che agisce istantaneamente. Pertanto potrà verificarsi che ci riferiremo al "calcolo portato avanti dall'oracolo" o simili: in tutti i casi tale computazione ausiliaria si assume richiedere un solo step.

Osservazione. Una macchina a oracolo può decidere qualunque linguaggio L in un numero di passi lineare (nella lunghezza dell'input): basta considerare la macchina che ha come oracle set proprio quel linguaggio L e non fa altro che copiare la stringa in input sul nastro d'oracolo e avere q_{yes} come unico stato accettante.

1.5 Richiami di complessità

Vediamo come il formalismo delle macchine di Turing ci permetta di definire con facilità le classi di complessità temporali e spaziali.
Mentre $\mathbb{N}$ indica il celebre insieme dei numeri naturali (a partire da 0), indichiamo con $\mathbb{N}^* \coloneqq \mathbb{N} \setminus \{0\}$.
In questo paragrafo a meno che diversamente specificato le funzioni saranno $\mathbb{N} \longrightarrow \mathbb{N}^*$ ed assumeremo che le macchine di Turing abbiano un solo nastro di lavoro, per snellire le definizioni.

Definizione 1.5.1. Sia f una funzione. Diciamo che una macchina di Turing M (deterministica o meno) ha **RUNNING TIME** f se esiste $c \in \mathbb{R}$ tale che per ogni stringa $w \in \mathcal{L}(M)$ (eccetto al più un numero finito di stringhe), M calcola w in non più di $c \cdot f(|w|)$ passi. Indichiamo con

$$DTIME(f)=\{L \mid \text{esiste una TM deterministica che decide } L \text{ con running time } f\}$$

$$NTIME(f)=\{L \mid \text{esiste una TM non deterministica che decide } L \text{ con running time } f\}$$

Definizione 1.5.2. Si chiama classe (temporale) **POLINOMIALE DETERMINISTICA** la seguente collezione di problemi:

$$P = \bigcup_{c \geq 1} DTIME(n^c)$$

Si chiama classe (temporale) **POLINOMIALE NON DETERMINISTICA** la seguente collezione di problemi:

$$NP = \bigcup_{c \geq 1} NTIME(n^c)$$

Molti celebri problemi sui grafi si trovano in NP (colorazione di grafi, cicli hamiltoniani, problema del commesso viaggiatore...), ma anche problemi di logica (soddisfacibilità di forme normali congiuntive, SAT) ed il loro studio è una prima interessante applicazione della teoria della complessità strutturale.

Osservazione. $P \subseteq NP$ poiché qualunque TM deterministica è in particolare non deterministica. Si ritiene che sia $P \subsetneq NP$ nonostante non esista una prova formale al riguardo. Tale problema aperto è uno dei sette celebri "Problemi del Millennio" per la risoluzione di ciascuno dei quali è istituito un premio da un milione di dollari. Viene comunemente detto problema $P \; vs \; NP$. Torneremo sulla questione dopo aver mostrato alcune proprietà della gerarchia polinomiale.

Definizione 1.5.3. Si definisce

$$co\text{-}NP \coloneqq \{L \mid \bar{L} \in NP\}$$

ed è abbastanza facile vedere che $P \subseteq co\text{-}NP$.

Definizione 1.5.4. Più in generale, data una classe di complessità $\mathcal{C}$, si definisce:

$$co\text{-}\mathcal{C} \coloneqq \{L \mid \bar{L} \in \mathcal{C}\}.$$

Definizione 1.5.5. Indichiamo con FP la classe di funzioni tra stringhe che possono essere calcolate da un trasduttore in tempo polinomiale.

Definizione 1.5.6. Sia f una funzione. Diciamo che una macchina di Turing M (deterministica o meno) ha **RUNNING SPACE** f se esiste $c \in \mathbb{R}$ tale che per ogni stringa $w \in \mathcal{L}(M)$ (eccetto al più un numero finito di stringhe), M non usa più di $c \cdot f(|w|)$ celle distinte nel nastro di lavoro durante il calcolo. Indichiamo con

$$DSPACE(f)=\{L \mid \text{esiste una TM deterministica che decide } L \text{ con running space } f\}$$

$$NSPACE(f)=\{L \mid \text{esiste una TM non deterministica che decide } L \text{ con running space } f\}$$

Definizione 1.5.7. Si chiama classe (spaziale) **POLINOMIALE DETERMINISTICA** la seguente collezione di problemi:

$$PSPACE \coloneqq \bigcup_{c \geq 1} DSPACE(n^c)$$

Osservazione. $P \subseteq PSPACE$. Infatti se una TM compie un numero di step polinomiale nella taglia dell'input, chiaramente essa potrà utilizzare al più un numero di celle ancora polinomiale nella taglia dell'input.
Un risultato classico della complessità strutturale è il seguente, che useremo e generalizzeremo nel prossimo capitolo (all'interno della Proposizione 3):

Lemma 3. $NP \subseteq PSPACE$.

Dimostrazione. Al termine di tutte le definizioni di questo capitolo, possiamo affemare che un linguaggio L è in NP se esiste una macchina non deterministica N_L tale che per ogni stringa $w \in L$, l'albero di computazione di N_L su w esibisce una foglia accettante a profondità polinomiale in $|w|$ e per ogni $w \notin L$, tutti i rami[2] dell'albero di computazione di N_L su w terminano con configurazioni non accettanti di profondità polinomiale in $|w|$. Inoltre si noti che ciascuna configurazione possibile di N_L su w ha taglia polinomiale in $|w|$. In particolare descrivere uno dei possibili rami di computazione di N_L su w (dalla confinguarazione iniziale a quella finale, passando per al più una quantità polinomiale di scelte binarie) genera una successione di configurazioni di lunghezza complessivamente polinomiale nella taglia dell'input.
La macchina deterministica M simulerà N_L sfruttando questo fatto ed esplorando l'albero di computazione di N_L su w ramo per ramo, cancellando di volta in volta il suo nastro e sfruttando dei contatori che tengano traccia della profondità e di quale dei possibili[3] rami sia esplorato. Siano dunque $L \in NP$ un linguaggio ed N_L una macchina che decide L con le caratteristiche appena descritte e dimostriamo che è possibile simulare N_L con una macchina deterministica M che usa un numero di celle distinte polinomiale.
Data una stringa w, $|w| = n$, sappiamo che esiste un polinomio $p(n)$ tale che la profondità della configurazione accettante di N_L su w è profonda $p(n)$ oppure tutti i rami dell'albero di computazione di N_L su w sono non accettanti ed hanno profondità minore o uguale di $p(n)$.
Si consideri la macchina deterministica M che calcola w nel modo seguente:

1. M tiene sempre sul nastro la configurazione iniziale ID_0 di N_L su w ed azzera il contatore;

[2]Per evitare fraintendimenti, ricordiamo che qui usiamo il termine "ramo" per indicare un qualunque "cammino radice-foglia"

[3]Si noti che il numero di possibili rami di calcolo di N_L su w è al più $\mathcal{O}(2^{p(n)})$ per un qualche polinomio $p(n)$, dove $n = |w|$. Tuttavia, il numero di rami in questione non è rilevante ai nostri fini poiché siamo interessati alle celle distinte utilizzate da M, non al tempo impiegato per la simulazione.

2. induttivamente, M legge l'ultima configurazione scritta sul suo nastro, simula una configurazione di N_L ad essa successiva e la scrive sul nastro;

3. M aggiorna il contatore aumentandolo di 1 (il contatore della profondità simulazione);

4. M ripete gli step 2 e 3 finché il contatore non raggiunge la quantità $p(n)$: se non può performare il punto 2 torna al punto 1), poiché si trova sicuramente su un ramo non accettante);

5. quando il contatore raggiunge $p(n)$, M valuta se la configurazione scritta sul nastro è accettante per N_L oppure no: in caso di risposta affermativa, M accetta la stringa w.

A meno di aggiungere un ulteriore contatore che permetta di numerare i rami e di simularne sempre di nuovi[4], possiamo costruire M in modo tale che quando M tale contatore superi la quantità $\mathcal{O}(p(n))$, essa rifiuti w. Una simile macchina M è sicuramente deterministica, decide L ed usa nel suo funzionamento un numero di celle non più che polinomiale nella taglia dell'input.
Pertanto, $NP \subseteq PSPACE$.

□

[4]Si noti come la taglia di un tale contatore è polinomiale in n.

Capitolo 2

La gerarchia polinomiale

Definiamo ora gli ultimi concetti chiave per comprendere e studiare la gerarchia polinomiale. Le dimostrazioni che seguiranno saranno talvolta discorsive con alcune verifiche lasciate al lettore.

2.1 Problemi completi e riduzioni

D'ora in avanti assumiamo che i linguaggi di cui trattiamo siano defniti su un alfabeto binario. È facile esercizio mostrare che tale assunzione non è restrittiva in quanto qualunque alfabeto $\Sigma = \{a_1, \cdots, a_n\}$ può essere codificato (lettera per lettera) in binario. In informatica esistono diversi tipi di riduzione: presentiamo qui i più utili al nostro scopo.

Definizione 2.1.1. Dati due linguaggi A e B, una **RIDUZIONE POLINOMIALE** da A a B è una funzione $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ tale che:

1. f è calcolabile da un trasduttore in tempo polinomiale ($f \in FP$);

2. per ogni stringa $w \in \{0,1\}^*$, $w \in A \iff f(w) \in B$, ossia f trasforma istanze del linguaggio di partenza in istanze del linguaggio di arrivo e non-istanze in non-istanze. In tal caso scriviamo $A \leq_p B$.

Definizione 2.1.2. Dati due linguaggi A e B, si dice che A è **TURING RIDUCIBILE** su B se esiste una macchina a oracolo $M^?$ che accetta A con oracle set B. In tal caso si scrive $A \leq^T B$.

Lemma 4 (Transitività delle riduzioni polinomiali)**.** Siano A, B, C linguaggi. Se $A \leq_p B$ e $B \leq_p C$, allora $A \leq_p C$.

Dimostrazione. Sia f_1 riduzione polinomiale da A a B calcolata dal trasduttore M_1 con running time il polinomio p_1. Analogamente, sia f_2 riduzione polinomiale da B a C calcolata dal trasduttore M_2 con running time il polinomio p_2. Consideriamo la funzione $f_2 \circ f_1$ e mostriamo che è riduzione polinomiale da A a C. Per ogni $w \in \{0,1\}^*$, $w \in A \iff f_2 \circ f_1(w) \in C$ per proprietà di f_1 e f_2. Inoltre, $f_2 \circ f_1$ è calcolabile in tempo polinomiale poichè il numero di step che performa (nella taglia del suo input w) è limitato a meno di una costante da un polinomio di grado pari al prodotto del grado di p_1 con quello di p_2, ancora per proprietà di f_1 ed f_2. □

Definizione 2.1.3. Sia $\mathcal{C}$ una classe di complessità e L un linguaggio. Si dice che L è **$\mathcal{C}$-HARD** se per ogni $A \in \mathcal{C}$ vale $A \leq_p L$. Diciamo che L è **$\mathcal{C}$-COMPLETO** se: $\begin{cases} L \text{ è } \mathcal{C}-\text{hard} \\ L \in \mathcal{C}. \end{cases}$

Corollario 1. Sia $\mathcal{C}$ una classe di complessità. Siano A un linguaggio $\mathcal{C}$-hard e B un linguaggio tale che $A \leq_p B$. Allora anche B è $\mathcal{C}$-hard.

Dimostrazione. Semplice verifica: sia $L \in \mathcal{C}$ un linguaggio. Per ipotesi, $L \leq_p A$ ed $A \leq_p B$: per la transitività delle riduzioni polinomiali anche $L \leq_p B$ e quindi B è $\mathcal{C}$-hard. □

Osservazione. I problemi citati per NP nel capitolo precedente (colorazione di grafi, cicli hamiltoniani, problema del commesso viaggiatore, SAT) sono tutti NP-completi. Storicamente la completezza è stata provata in modo diretto solo per il problema SAT, e poi il corollario precedente e le riduzioni polinomiali hanno esteso la hardness a tutti gli altri.
Più in generale esistono problemi completi anche per $PSPACE$ e le altre classi che abbiamo nominato (e nomineremo).

Definizione 2.1.4. Data una classe di complessità $\mathcal{C}$ e una collezione qualunque di linguaggi D, indichiamo con $\mathcal{C}^D$ la classe di linguaggi decidibili da macchine a oracolo $M^?$ che hanno complessità $\mathcal{C}$ e oracle set in D.

2.2 La gerarchia polinomiale

La **GERARCHIA POLINOMIALE** è la struttura formata dalle classi Σ_k^p, Π_k^p e Δ_k^p per ogni $k \geq 0$, dove:

$$\begin{cases} \Sigma_0^p = \Pi_0^p = \Delta_0^p = P \\ \Sigma_{k+1}^p = NP^{\Sigma_k^p} \text{ per ogni } k \geq 0 \\ \Pi_{k+1}^p = (co\text{-}NP)^{\Sigma_k^p} \text{ per ogni } k \geq 0 \\ \Delta_{k+1}^p = P^{\Sigma_k^p} \text{ per ogni } k \geq 0. \end{cases}$$

Si definisce poi

$$PH = \bigcup_{k \geq 0} \Sigma_k^p$$

In letteratura si è soliti usare l'apice p per distinguere la gerarchia polinomiale da altre gerarchie (quella esponenziale ad esempio).

Osservazione. Si presti attenzione alla differenza tra $(co\text{-}NP)^{\Sigma_k^p}$ e $co\text{-}(NP^{\Sigma_k^p})$: le parentesi svolgono un ruolo fondamentale per distinguere classi diverse. In ogni caso, la seguente Proposizione tra le altre cose caratterizza Π_k^p identificandolo con $co\text{-}\Sigma_k^p$ e snellendo così la notazione.

Proposizione 1. Per ogni $k \geq 0$ valgono le seguenti:

1. $\Delta_1^p = P$;
2. $\Pi_k^p = co\text{-}\Sigma_k^p$;
3. $\Sigma_{k+1}^p = NP^{\Pi_k^p}$;
4. $\Delta_{k+1}^p = P^{\Pi_k^p}$;
5. $\Sigma_{k+1}^p = NP^{\Delta_{k+1}^p}$;
6. $\Pi_{k+1}^p = (co\text{-}NP)^{\Delta_{k+1}^p}$

Dimostrazione. 1) Per definizione, $\Delta_1^p = P^P = \{L \mid L \text{ può essere deciso da una TM deterministica polinomiale con oracolo in } P\}$. La transitività delle riduzioni polinomiali mostra che avere un oracolo (cioè una subroutine istantanea) per decidere stringhe che possono essere decise in tempo polinomiale deterministico all'interno di una macchina polinomiale deterministica non dà un vero e proprio vantaggio in termini di numero di passi (esso rimarrà in ogni caso polinomiale), da cui $P^P = P$ e quindi possiamo concludere.
2) Il caso $k = 0$ è facile perchè $co\text{-}P = P$ (banalmente il complemento di un linguaggio si decide con la stessa TM che decide il linguaggio, scambiando gli stati accettanti con quelli non accettanti) . D'altra parte, $L \in \Pi_k^p \iff$

$$\iff L \in (co\text{-}NP)^{\Sigma_{k-1}^p} \overset{(*)}{\iff} \bar{L} \in NP^{\Sigma_{k-1}^p} \iff \bar{L} \in \Sigma_k^p \iff L \in co\text{-}\Sigma_k^p,$$

dove in tutte le equivalenze abbiamo usato le definizioni, tranne che in $(*)$. Quest'ultima è però giustificata dal fatto che $L \in co\text{-}NP \iff \bar{L} \in NP$ e l'accesso all'oracolo non cambia il ragionamento.
3) Siano $A \in \Sigma^p_{k+1}$ e M^L una TM polinomiale non deterministica che decide A con $L \in \Sigma^p_k$. Consideriamo la macchina $\tilde{\mathrm{M}}^{\bar{L}}$ (con oracle set $\bar{L}$) in cui $\tilde{\mathrm{M}}$ è identica a M ma con gli stati q_{yes} e q_{no} invertiti rispetto ad essa. $\tilde{\mathrm{M}}$ è ancora polinomiale non deterministica , $\tilde{\mathrm{M}}^{\bar{L}}$ decide A e ciò sancisce che $A \in NP^{\Pi^p_k}$. Il viceversa è analogo.
4) Come al punto 3).
5) $\Sigma^p_{k+1} \subseteq NP^{\Delta^p_{k+1}}$ poiché $\Sigma^p_k \subseteq \Delta^p_{k+1}$. Viceversa, siano L un linguaggio in Δ^p_{k+1} e M_0 una qualunque TM deterministica polinomiale che accetta L con un oracolo in Σ^p_k. Per qualunque macchina M_1 (non deterministica e polinomiale) che usa L come oracolo, si costruisca la macchina M_2 la quale:

- è programmata come M_1 ogni volta che M_1 non si trova nello stato $q_?$;
- ogni volta che M_1 transisce nello stato $q_?$ ed ha sul nastro d'oracolo una certa stringa v, M_2 simula passo dopo passo la subroutine deterministica M_0 su v e, in accordo con la risposta ottenuta da questa simulazione, si sposta nello stato q_{yes} o q_{no}, cancella v dal nastro e prosegue la computazione come avrebbe fatto M_1.

Dato che M_1 era non deterministica e polinomiale, anche M_2 lo è. Tuttavia l'oracolo di M_2 è in Σ^p_k, non più in Δ^p_{k+1}. Questo prova che $NP^{\Delta^p_{k+1}} \subseteq NP^{\Sigma^p_k} = \Sigma^p_{k+1}$, cioè l'inclusione che rimaneva.
6) Come al punto 5). □

Proposizione 2. Per ogni $k \geq 0$ valgono le seguenti:

1. Δ^p_k è chiuso sotto complementazione;
2. $P^{\Delta^p_k} = \Delta^p_k$, cioè Δ^p_k è chiuso sotto polinomiale Turing-riducibilità;
3. $\Sigma^p_k \cup \Pi^p_k \subseteq \Delta^p_{k+1}$;
4. $\Delta^p_k \subseteq \Sigma^p_k \cap \Pi^p_k$;
5. tutte le classi della gerarchia polinomiale sono chiuse sotto riduzione polinomiale, ossia se $\mathcal{C} \in \{\Sigma^p_k, \Pi^p_k, \Delta^p_k \mid k \geq 0\}$ e B è un linguaggio di $\mathcal{C}$, si ha sempre: $A \leq_p B \Rightarrow A \in \mathcal{C}$;
6. se $\Sigma^p_k \subseteq \Pi^p_k$ oppure $\Pi^p_k \subseteq \Sigma^p_k$ allora $\Sigma^p_k = \Pi^p_k$.

Dimostrazione. 1) Vediamo il caso $k \neq 0$: il caso $k = 0$ segue lo stesso ragionamento ma senza oracoli coinvolti. Siano $L \in \Delta_k^p$ e M macchina deterministica polinomiale che accetta L con oracolo in Σ_{k-1}^p. Consideriamo la macchina $\tilde{\mathrm{M}}$ ottenuta scambiando gli stati accettanti di M con quelli non accettanti e per il resto uguale a M. Essa accetta $\bar{L}$ ed è ancora polinomiale deterministica, da cui $\bar{L} \in \Delta_k^p$.
2) Per induzione su k: il caso base $k = 0$ è già stato visto nella Proposizione 1. Per quanto riguarda il passo induttivo, basta mostrare che $P^{\Delta_k^p} \subseteq \Delta_k^p$ poiché l'altra inclusione è ovvia. Sia $A \in P^{\Sigma_{k-1}^p}$ oracolo per una TM M_0 deterministica polinomiale che accetta un certo linguaggio di $P^{\Delta_k^p}$. Sia M_1 macchina polinomiale deterministica che accetta A usando un oracolo in Σ_{k-1}^p. Si consideri la macchina M_2 che simula M_0 ogni volta che M_0 non si trova nello stato $q_?$ e che simula M_1 quando M_0 fa domande all'oracolo: M_2 sarà polinomiale deterministica perché M_0 ed M_1 lo sono; inoltre M_2 ha lo stesso oracolo di M_1, che è in Σ_{k-1}^p. Quindi ogni linguaggio di $P^{\Delta_k^p}$ è anche in Δ_k^p, il che conclude. Quest'ultimo ragionamento del passo induttivo è simile a quello proposto nel punto 5) della Proposizione 1, ma più semplice perché le macchine in questione sono deterministiche.
3) $\Sigma_k^p \subseteq \Delta_{k+1}^p$ direttamente per definizione. D'altra parte, sia $L \in \Pi_k^p$: allora $\bar{L} \in \Sigma_k^p \subseteq \Delta_{k+1}^p$ e dato che vale la 1), anche $L \in \Delta_{k+1}^p$.
4) $\Delta_k^p \subseteq \Sigma_k^p$ chiaro per definizione. Inoltre, sia $L \in \Delta_k^p$: allora anche $\bar{L} \in \Delta_k^p$ da cui $\bar{L} \in \Sigma_k^p \iff L \in co\text{-}\Sigma_k^p = \Pi_k^p$, il che prova che $\Delta_k^p \subseteq \Pi_k^p$.
5) Dimostriamo solo il caso $\mathcal{C} = \Sigma_k^p$. Sia f riduzione polinomiale da A a B calcolata dal trasduttore M_0, con $B \in \Sigma_k^p$. Sia M_1 macchina di Turing polinomiale non deterministica con oracolo in Σ_{k-1}^p che accetta B. La macchina che simula prima M_0 e poi è identica a M_1 è polinomiale non deterministica, sfrutta un oracolo in Σ_{k-1}^p e accetta A. Pertanto $A \in \Sigma_k^p$.
6) Facile: vediamo solo nel caso in cui $\Sigma_k^p \subseteq \Pi_k^p$. Per ogni linguaggio L, si ha $L \in \Pi_k^p \iff \bar{L} \in \Sigma_k^p$ da cui $\bar{L} \in \Pi_k^p$ e quindi anche $L \in \Sigma_k^p$. Cioè vale $\Pi_k^p \subseteq \Sigma_k^p$, che è l'altro contenimento. □

Osservazione. Dalla proposizione precedente segue facilmente che

$$PH = \bigcup_{k \geq 0} \Sigma_k^p = \bigcup_{k \geq 0} \Pi_k^p = \bigcup_{k \geq 0} \Delta_k^p.$$

Inoltre valgono i seguenti contenimenti:

$$P \subseteq \Sigma_k^p \cup \Pi_k^p \subseteq \Delta_{k+1}^p \subseteq \Sigma_{k+1}^p \cap \Pi_{k+1}^p \subseteq \cdots \subseteq PH$$

Non è chiaro se questi contenimenti siano stretti o meno, così come se la gerarchia polinomiale sia composta da infinite classi oppure no: rimangono problemi aperti di cui tratteremo brevemente tra poco (si osservi anche la Figura 2.1).

Proposizione 3. $PH \subseteq PSPACE$.

Dimostrazione. Mostriamo per induzione su k che $\Sigma_k^p \subseteq PSPACE$: da ciò segue facilmente la tesi. Il caso base è già stato visto nella parte finale del capitolo precedente.
Supponiamo ora che $\Sigma_k^p \subseteq PSPACE$: allora chiaramente avremo anche $\Sigma_{k+1}^p = NP^{\Sigma_k^p} \subseteq NP^{PSPACE}$. Per concludere ci basta dimostrare che $NP^{PSPACE} \subseteq PSPACE$. Tuttavia tale contenimento è facile conseguenza della dimostrazione del Lemma 3. Infatti, dato un linguaggio L accettato da una macchina polinomiale non deterministica N - che sfrutta un oracolo $A \in PSPACE$ - esso può essere deciso da una macchina M deterministica che simula N nel modo descritto nel Lemma 3 ogni qualvolta N non si trova nello stato $q_?$ e che simula il decisore per A per avere le risposte dell'oracolo ogni volta che N si trova nello stato $q_?$. Una tale M è deterministica poiché il decisore di A lo è, accetta L e usa un numero di celle al più polinomiale nella taglia dell'input in quanto $A \in PSPACE$ (quindi il suo decisore di volta in volta usa una quantità di celle ancora non più che polinomiale nella taglia dell'input). □

2.3 Relazioni con altre classi

Vediamo infine come si può utilizzare la gerarchia polinomiale per dedurre alcune proprietà sorprendenti, legate anche alla questione P *vs* NP.

Definizione 2.3.1. Sia $i \geq 1$. Diciamo che la gerarchia polinomiale **COLLASSA AL LIVELLO i-ESIMO** se:

$$\Sigma_{i+1}^p = \Sigma_i^p.$$

Si nota che in tal caso:

$$\Sigma_{i+2}^p = NP^{\Sigma_{i+1}^p} = NP^{\Sigma_i^p} = \Sigma_{i+1}^p = \Sigma_i^p$$

e che quindi anche $\Sigma_{i+k}^p = \Sigma_i^p$ per ogni $k \geq 0$.

Proposizione 4. Se la gerarchia collassa al livello i-esimo allora per ogni $k \geq 0$:

1. $\Pi_{i+k}^p = \Pi_i^p$;
2. $\Delta_{i+k}^p = \Delta_i^p$;

3. $\Sigma^p_{i+k} = \Pi^p_{i+k} = \Delta^p_{i+k} = \Sigma^p_i$, cioè tutte le classi superiori collassano su Σ^p_i.

Dimostrazione. 1) $\Pi^p_{i+k} = co\text{-}\Sigma^p_{i+k} = co\text{-}\Sigma^p_i = \Pi^p_i$ per ogni $k \geq 0$.
2) Per induzione su k, con caso base ovvio. Sia $k \in \mathbb{N}^*$ fissato e supponiamo che $\Delta^p_{i+j} = \Delta^p_i$ per ogni $j < k$. Si ha che:

$$\Delta^p_{i+k} = P^{\Sigma^p_{i+k-1}} = P^{\Sigma^p_i} = \Delta^p_{i+1} = \Delta^p_i$$

dove abbiamo usato il collassamento al livello i-esimo e l'ipotesi induttiva, rispettivamente, nelle ultime uguaglianze.
3) Procediamo per gradi. Dalle proprietà dei contenimenti:

$$\Sigma^p_i \cup \Pi^p_i \subseteq \Delta^p_{i+1} \subseteq \Sigma^p_{i+1} \cap \Pi^p_{i+1} = \Sigma^p_i \cap \Pi^p_i$$

da cui ricaviamo che $\Pi^p_i = \Sigma^p_i$. Pertanto anche $\Sigma^p_{i+k} = \Pi^p_{i+k} = \Sigma^p_i$ per ogni $k \geq 0$. D'altra parte vale $\Delta^p_{i+1} \subseteq \Sigma^p_{i+1} \cap \Pi^p_{i+1} = \Sigma^p_i$ e quindi anche $\Delta^p_{i+1} = \Sigma^p_i$ e usando il punto 2) possiamo concludere. □

Osservazione. Il prossimo risultato rappresenta una sorta di "implicazione opposta" alla Proposizione 4 e la sua dimostrazione supera gli obiettivi della trattazione, in quanto sfrutta la caratterizzazione della gerarchia polinomiale attraverso TM alternanti.

Lemma 5. Se per qualche $i \geq 1$ vale $\Sigma^p_i = \Pi^p_i$ allora la gerarchia collassa al livello i-esimo.

Corollario 2. Se esiste un k tale che $\Sigma^p_k \neq P$ allora $P \neq NP$.

Dimostrazione. Dimostriamo che se $P = NP$ allora per ogni k vale $\Sigma^p_k = P$. Se $P = NP$, allora $\Sigma^p_1 = NP^P = P^P = P = \Sigma^p_0$ e $\Pi^p_1 = co\text{-}\Sigma^p_1 = co\text{-}P = P$ da cui vale il Lemma 5 e possiamo concludere. □

Corollario 3. Si verificano due casi: o per tutti i $k \geq 0$ vale $\Sigma^p_k \neq \Sigma^p_{k+1}$ oppure la gerarchia polinomiale collassa a un qualche livello.

Dimostrazione. Già visto di fatto: si dimostra che se non vale uno scenario vale necessariamente l'altro, il che è facile conseguenza dei corollari visti finora. □

Corollario 4. Se $PH = PSPACE$ allora la gerarchia polinomiale collassa a un qualche livello.

Dimostrazione. Sia L un problema $PSPACE$-completo. Dato che $PH = PSPACE$, $L \in PH$ e quindi esiste un certo $k \geq 0$ tale che $L \in \Sigma^p_k$. Dato che L è completo, qualunque linguaggio di $PSPACE$ si può ridurre polinomialmente a L, in Σ^p_k: in particolare tutti i problemi di Π^p_k si possono ridurre ad L;

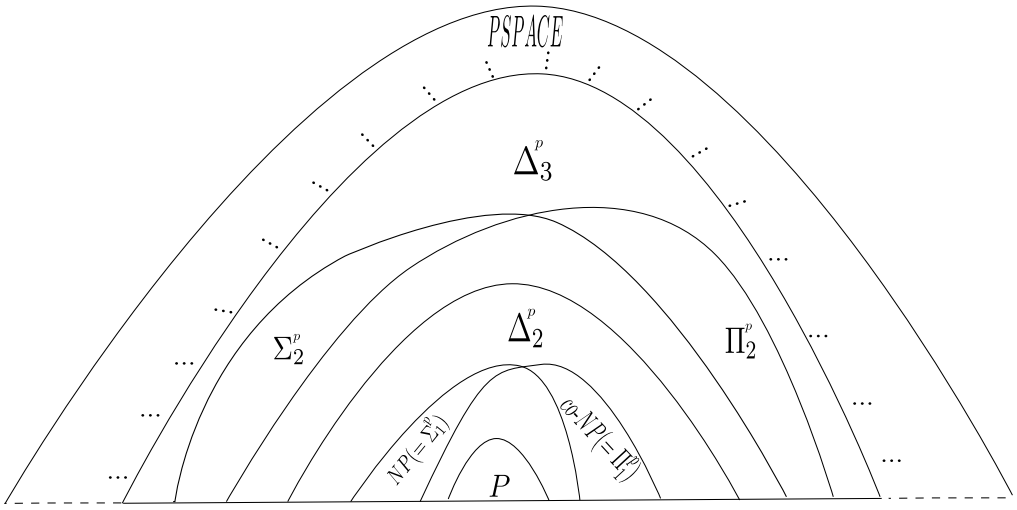

Figura 2.1: Rappresentazione visiva dei contenimenti della gerarchia polinomiale

dato che Σ_k^p è chiuso sotto riduzioni polinomiali, $\Pi_k^p \subseteq \Sigma_k^p$ da cui $\Pi_k^p = \Sigma_k^p$ e la gerarchia collassa al livello k. □

Corollario 5. Se esiste un linguaggio L che sia PH-completo, allora la gerarchia collassa a qualche livello.

Dimostrazione. Dato che $L \in PH$, esiste $k \geq 0$ tale che $L \in \Sigma_k^p$ e si ragiona come nel Corollario 4. □

Osservazione. I corollari di questa sezione hanno conseguenze dirette sul problema ***P vs NP*** ed anzi sono di fatto implicazioni che lo risolvono direttamente. La motivazione risiede nelle Proposizioni 2 e 3: la gerarchia polinomiale infatti costituisce una serie di classi intermedie tra P, NP e $PSPACE$ - come un vero e proprio cuscinetto senza il comfort del quale gli strati P ed $NP = \Sigma_1^p$ vanno a coincidere. Si guardi la **Figura 2.1** per avere un'intuizione visiva.

2.4 Problemi completi per la gerarchia polinomiale

Presentiamo ora una serie di problemi che vengono dal mondo della logica booleana e che si può dimostrare essere completi per la gerarchia polinomiale.

Definizione 2.4.1. Una **VARIABILE BOOLEANA** è un simbolo formale cui possiamo associare un solo valore di verità 0 oppure 1 ("falso" o "vero", rispettivamente).

Definizione 2.4.2. Sia X un insieme numerabile di variabili booleane. La famiglia delle **FORMULE BOOLEANE** su X si definisce induttivamente come:

1. 0 e 1 sono formule booleane dette *costanti booleane*;
2. per ogni variabile booleana $x \in X$, x è una formula booleana;
3. se F_1 e F_2 sono formule booleane, allora $(F_1 \wedge F_2), (F_1 \vee F_2)$ e $\neg(F_1)$ sono formule booleane;

Poi, la classe delle **FORMULE BOOLEANE QUANTIFICATE** su X è definita da due ulteriori regole:

4. qualunque formula booleana è una formula booleana quantificata;
5. se $x \in X$ e F è una formula booleana quantificata, allora $\exists xF$ e $\forall xF$ sono formule booleane quantificate

I simboli $\wedge, \vee, \neg$ sono detti *connettivi booleani* e svolgono il ruolo rispettivamente di "e", "o" e "non", che si usano comunemente, ed in tal modo si pronunciano. Invece i simboli $\forall$, $\exists$ sono detti rispettivamente *quantificatore universale* e *quantificatore esistenziale*. Spesso ometteremo le parentesi tonde quando scriveremo formule booleane.

Definizione 2.4.3. Data una formula booleana quantificata, una variabile in essa presente ma che non è accompagnata da un quantificatore viene detta **LIBERA**.

Definizione 2.4.4. Sia F una formula booleana quantificata su X, $x \in X$ e $a \in X \cup \{0,1\}$. Indicheremo con $F|_{x=a}$ la formula ottenuta sostituendo con a ogni occorrenza libera di x in F.

Definizione 2.4.5. Un **ASSEGNAMENTO** booleano è una funzione $w : X \longrightarrow \{0,1\}$. Data una formula booleana quantificata F e un assegnamento w, il **VALORE DI VERITÀ** di F, indicato $w(F)$, è così definito:

1. $w(F) = 0$ se $F = 0$;
2. $w(F) = 1$ se $F = 1$;
3. $w(F) = w(x)$ se $F = x$, con $x \in X$;
4. $w(F) = \max\{w(F_1), w(F_2)\}$ se $F = (F_1 \vee F_2)$;

5. $w(F) = \min\{w(F_1), w(F_2)\}$ se $F = (F_1 \wedge F_2)$;

6. $w(F) = 1 - w(F_1)$ se $F = \neg F_1$;

7. $w(F) = \max\{w(F_1|_{x=0}), w(F_1|_{x=1})\}$ se $F = \exists x F_1$;

8. $w(F) = \min\{w(F_1|_{x=0}), w(F_1|_{x=1})\}$ se $F = \forall x F_1$;

Definizione 2.4.6. Data una formula F ed un assegnamento w, diciamo che w **SODDISFA** F se $w(F) = 1$. Una formula si dice **SODDISFACIBILE** se esiste un assegnamento che la soddisfa.

Definizione 2.4.7. Date due formule booleane quantificate F_1 ed F_2, diciamo che esse sono **EQUIVALENTI** se per qualunque assegnamento w si ha $w(F_1) = w(F_2)$.

Definizione 2.4.8. Un **LETTERALE** è una formula booleana data da una variabile booleana o dalla negazione di una variabile booleana. Una **CLAUSOLA** è una disgiunzione di letterali.

Quindi una clausola ha la forma $C = l_1 \vee l_2 \vee \cdots \vee l_n$ dove l_i è una variabile booleana x_k oppure la sua negazione $\neg x_k$.

Definizione 2.4.9. Una formula booleana è in **FORMA NORMALE CONGIUNTIVA** (CNF) se è congiunzione di clausole.

Esempio. La formula

$$\varphi = (x_1 \vee \neg x_3) \wedge x_2$$

è una formula booleana CNF costituita da due clausole, la prima data da due letterali, la seconda composta da solo uno di essi. Essa è soddisfacibile, ad esempio tramite l'assegnamento che associa ad x_1 e ad x_2 il valore 1 (e agisce arbitrariamente sulle altre variabili di X).

È facile vedere che qualunque formula booleana di variabili $x_1, \ldots, x_n$ può essere codificata in linguaggio binario: basta prendere l'alfabeto

$$\Sigma = \{x_1, \ldots, x_n, \neg, \vee, \wedge, (, \text{“}vero\text{”}, \text{“}falso\text{”},)\}$$

e mapparlo nel linguaggio binario, lettera per lettera. Il linguaggio di tutte le formule booleane soddisfacibili è indicato SAT ed è il primo esempio di linguaggio NP-completo storicamente trovato. In simboli:

$$\text{SAT}=\{\varphi \mid \varphi \text{ è una formula booleana soddisfacibile}\}.$$

Indicheremo anche:

$$\text{CNF}=\{\varphi \mid \varphi \text{ è una formula booleana CNF soddisfacibile}\}.$$
$$\text{3CNF}=\{\varphi \mid \varphi \text{ è una CNF e ogni sua clausola contiene al più 3 letterali}\}.$$

Problemi completi per le varie classi della gerarchia polinomiale saranno generalizzazioni di SAT, in cui sono presenti più "set" di variabili booleane quantificate.

Si considerino k famiglie diverse di variabili booleane:

$$(x_{1,1},\dots,x_{1,n_1}),\dots,(x_{k,1},\dots,x_{k,n_k});$$

le denoteremo rispettivamente con i vettori $\mathbf{x_1},\dots,\mathbf{x_k}$.

Cominciamo con problemi completi per le classi Σ_k^p, con $k \geq 1$.
Fissiamo $k \geq 1$. Un problema Σ_k^p-completo è dato dallo stabilire quali formule booleane nelle variabili $\mathbf{x_1},\dots,\mathbf{x_k}$ a quantificatori alternati siano soddisfacibili:

$$QSAT_k = \{\varphi = \varphi(\mathbf{x_1},\dots,\mathbf{x_k}) \mid \varphi \text{ è una formula booleana ed } \exists\mathbf{x_1}\forall\mathbf{x_2}\dots Q_k\mathbf{x_k}\varphi(\mathbf{x_1},\dots,\mathbf{x_k}) \text{ è soddisfacibile}\}$$

$$\text{dove } Q_k = \begin{cases} \exists \text{ se } k \text{ dispari} \\ \forall \text{ se } k \text{ pari} \end{cases}$$

Ad esempio $QSAT_1 = \exists SAT = SAT$ oppure $QSAT_2 = \exists\forall SAT$ e così via...

Esempio. Eliminando i doppi indici, nel caso $k = 2$ possiamo considerare la formula $\varphi = \varphi(x_1, x_2, y_1, y_2, y_3)$ data da:

$$\varphi = (x_1 \vee y_1 \vee \neg y_3) \wedge (\neg x_2 \vee y_2) \wedge x_1 \wedge \neg x_2.$$

Essa è un'istanza positiva di $QSAT_2$: infatti assegnando a x_1 il valore 1 e ad x_2 il valore 0, comunque siano assegnate le variabili y_1, y_2, y_3, si otterrà sempre che φ è soddisfatta. Si noti anche che φ è una 3CNF.

Problemi Π_k^p-completi sono molto simili: si tratta sempre di quantificare formule booleane in maniera alternata ma a partire da $\forall$.

$$NQSAT_k = \{\varphi = \varphi(\mathbf{x_1},\dots,\mathbf{x_k}) \mid \varphi \text{ è una formula booleana e } \forall\mathbf{x_1}\exists\mathbf{x_2}\dots Q_k\mathbf{x_k}\varphi(\mathbf{x_1},\dots,\mathbf{x_k}) \text{ è soddisfacibile}\}$$

$$\text{dove } Q_k = \begin{cases} \exists \text{ se } k \text{ pari} \\ \forall \text{ se } k \text{ dispari} \end{cases}$$

Ad esempio $NQSAT_1$=\{formule booleane soddisfatte per qualunque assegnamento, cioè quelle che sono dette tautologie\}, $NQSAT_2 = \forall\exists SAT$ e così via...

Problemi Δ_k^p-completi sono invece di natura diversa, in quanto si vanno a collocare "a metà" tra i problemi precedenti; ne incontreremo uno molto famoso nella sezione 3.6.

Capitolo 3

I giochi cooperativi

3.1 Introduzione alla teoria dei giochi

La ***teoria dei giochi*** è una disciplina che studia modelli matematici di interazione strategica tra agenti intelligenti. Nata negli anni Quaranta, si è sviluppata soprattutto in campo economico e politico, oltre che matematico. Nelle prossime pagine cercheremo di dare più spessore alla definizione e introdurremo concetti che ci saranno utili nel resto della trattazione.

La teoria dei giochi è molto vasta ma ha alcuni elementi comuni:

1. ogni ***gioco*** coinvolge un numero finito N di giocatori o individui, i quali spesso non sono persone ma entità qualunque (aziende, corporazioni, ma anche Stati e quindi talvolta ci riferiremo ad essi con il nome di "agenti") che assumeremo sempre ***intelligenti*** (o razionali), cioè i quali non solo conoscono tutte le regole del gioco, ma soprattutto perseguono l'obiettivo di vincere, o comunque ottenere il maggior vantaggio (o pay-off) possibile;

2. la mossa (o l'insieme di mosse) che un individuo intende eseguire viene chiamata ***strategia***: in dipendenza delle strategie adottate dai vari giocatori, ciascuno di essi riceve un pay-off, che noi assumeremo un numero reale; esso sarà positivo se la strategia seguita dall'agente lo avrà in qualche misura avvicinato alla vittoria e negativo in caso contrario;

3. matematicamente, il pay-off per ciascun giocatore sarà l'output di una funzione, la ***funzione di pay-off***;

4. il risultato del gioco sarà completamente determinato dalla sequenza delle strategie adottate dai vari individui.

Esistono moltissimi tipi di giochi, ciascuno modellizzato in maniera differente. Qui ci concentreremo sui *"giochi cooperativi ad utilità trasferibile"*.

Definizione 3.1.1. Un **GIOCO COOPERATIVO** è un gioco in cui agli individui è permesso (ed anzi spesso consigliato) associarsi in coalizioni per migliorare il proprio pay-off, rispetto che agendo in autonomia. Si dice che un gioco cooperativo è ad **UTILITÀ TRASFERIBILE** (TU) se i beni ottenuti in queste coalizioni possono essere distribuiti liberamente tra i vari giocatori, senza alcuna restrizione.

D'ora in avanti quando useremo il termine "gioco" intenderemo sempre "gioco cooperativo ad utilità trasferibile".
Un gioco di questo tipo può essere modellizzato come una coppia $\mathcal{G} = (N, \nu)$, dove N è un insieme finito di giocatori e ν è una funzione (talvolta parziale) $\nu : \mathcal{P}(N) \longrightarrow \mathbb{R}$ che associa a possibili coalizioni di individui $S \subseteq N$ una certa quantità $\nu(S) \in \mathbb{R}$, che i giocatori in S ottengono in forza della loro collaborazione.

Dato un gioco $\mathcal{G} = (N, \nu)$, la teoria dei giochi cooperativi TU si è sviluppata alla ricerca di vettori di pay-off (o **allocazioni**) $(x_i)_{i \in N}$ che specificano la distribuzione del valore totale a ciascun giocatore in modo che vengano rispecchiati in qualche misura concetti di equità o giustizia (e che così possano essere il più possibile accontentati i giocatori, in relazione al "potere contrattuale" di ognuno).

Definizione 3.1.2. Dato un gioco $\mathcal{G} = (N, \nu)$ ed un'allocazione $(x_i)_{i \in N}$, si dice che essa è **EFFICIENTE** se

$$\sum_{i \in N} x_i = \nu(N).$$

Definizione 3.1.3. Dato un gioco $\mathcal{G} = (N, \nu)$ ed un'allocazione $(x_i)_{i \in N}$, diciamo che essa è **INDIVIDUALMENTE RAZIONALE** se $x_i \geq \nu(\{i\})$ per ogni $i \in N$.

Denoteremo con $X(\mathcal{G})$ l'insieme di tutte le possibili allocazioni efficienti e individualmente razionali di un gioco $\mathcal{G}$.
Chiaramente, $X(\mathcal{G})$ è un insieme infinito e per questo motivo sono stati definiti diversi ***concetti di soluzione*** per un gioco $\mathcal{G}$, come le allocazioni che abbiano le caratteristiche per soddisfare più giocatori possibili.
Incontreremo alcuni concetti di soluzione nelle prossime sezioni.

Osservazione. Si noti come nella rappresentazione proposta dei giochi cooperativi, il valore reale $\nu = \nu(S)$ non dipenda dalla strategia seguita dai giocatori,

ma solo dalle coalizioni tra di essi. Questo modello rende lo studio dei giochi cooperativi assai diverso da quello dei giochi strategici (senza cooperazioni, in cui il pay-off sarà funzione delle strategie seguite dai vari giocatori e che dunque si è soliti studiare per mezzo di grafi etichettati con mosse e contromosse).

3.2 Rappresentazione delle informazioni e giochi compatti

Dal punto di vista informatico ci si è presto resi conto di un problema non banale, legato alla rappresentazione della funzione di pay-off ν per un gioco $\mathcal{G}$ in un computer. Infatti, operativamente listare i valori $\{\nu(S) \mid S \subseteq N\}$ è esponenziale in $|N|$, dato che $|\mathcal{P}(N)| = 2^{|N|}$. Ne segue che la rappresentazione di un gioco $\mathcal{G}$ "esplode" all'aumentare del numero di giocatori e ciò impedisce di trovare algoritmi veloci per la ricerca di allocazioni ottimali: effettivamente, come abbiamo visto, in complessità strutturale si lega sempre l'esecuzione di un algoritmo alla taglia del suo input e se quest'ultima è enorme, anche algoritmi polinomiali risultano computazionalmente non sostenibili. In questo contesto si è quindi cercato di trovare modi per rappresentare giochi e funzioni ν in maniera più concisa: giochi cooperativi in cui metodi per ottenere ciò sono stati trovati sono detti ***giochi compatti***. Si noti come non tutti i giochi cooperativi sono compatti, ma lo studio di rappresentazioni di giochi è in continuo sviluppo.

Nel proseguimento ci concentreremo su giochi compatti che ammettano un tipo di rappresentazione con grafi pesati, cioè su giochi ***a grafo***.

Definizione 3.2.1. Dato un gioco $\mathcal{G} = (N, \nu)$ a grafo, le ricchezze guadagnate da coalizioni in un insieme di N individui sono definite basandosi su un grafo pesato non orientato $G = ((N, E), w)$ i cui vertici sono i giocatori N del gioco, esiste un arco in E che collega due giocatori a e b di N se e solo se è fornito il valore $\nu(\{a, b\})$ e la lista $w : E \longrightarrow \mathbb{R}$ associa ad ogni arco $e \in E$ il suo peso, dato dal valore reale $\nu(\{\text{vertici di e}\})$. La ricchezza generata dalla generica coalizione $S \subseteq N$ è definita come la somma dei pesi degli archi i cui vertici sono in S.
In un gioco a grafo, il peso associato alla coalizione di un solo giocatore è 0 a meno che non sia fornito un arco pesato che entra ed esce dal nodo chiamato come quel giocatore.

Esempio. Si consideri il gioco a grafo rappresentato in **Figura 3.1** con giocatori $\{a, b, c, d\}$. È semplice vedere che la coalizione $\{a, b, c\}$ guadagna una ricchezza $\nu(\{a, b, c\}) = 2+2+1 = 5$, mentre la coalizione $\{a, b, d\}$ guadagna una ricchezza $\nu(\{a, b, d\}) = 2 + 3 - 1 = 4$. Si noti anche come il grafo codifichi 2^4 possibili

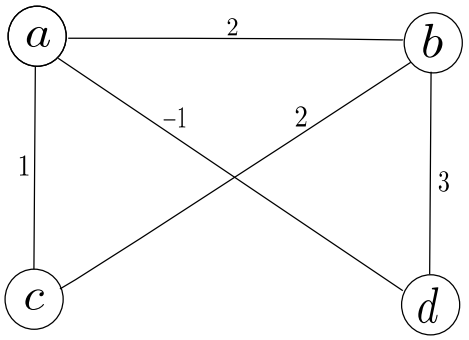

Figura 3.1: un esempio di gioco a grafo

coalizioni con 5 soli pesi: più in generale questa rappresentazione permette di usare $\mathcal{O}(n^2)$ pesi per 2^n coalizioni differenti.

Osservazione. Nonostante possa sembrare assai efficace, questa rappresentazione di giochi a grafo nasconde non pochi tranelli. Uno di questi è la sua limitata espressività, dato che non è difficile mostrare che essa non permette di descrivere giochi in cui ad esempio gruppi di giocatori S hanno guadagno 1 se e solo se $|S| > \frac{|N|}{2}$.

3.3 Il Bargaining Set

Vediamo ora un famoso concetto di soluzione per giochi cooperativi TU dato dall'insieme delle allocazioni in cui nessun giocatore possa "obiettare" alcunchè riguardo alla distribuzione delle ricchezze.

Definizione 3.3.1. Sia $\mathcal{G} = (N, \nu)$ un gioco e $S \subseteq N$ una coalizione. Indichiamo con $\mathbb{R}^S$ lo spazio vettoriale reale $|S|$-dimensionale, le cui coordinate sono etichettate con gli elementi di S (dato un vettore $x \in \mathbb{R}^S$, x_i denota la componente associata al giocatore $i \in S$). Un vettore $x \in \mathbb{R}^S$ viene detto **S-FATTIBILE** se:

$$\sum_{i \in S} x_i = \nu(S).$$

Denoteremo spesso con $x(S)$ il membro di sinistra dell'equazione precedente.

Definizione 3.3.2. Sia $\mathcal{G} = (N, \nu)$ un gioco, $x \in X(\mathcal{G})$ un'allocazione e sia $S \subseteq N$ una coalizione. Sia $y \in \mathbb{R}^S$ un vettore di pay-off per giocatori di S che sia S-fattibile. Diciamo che la coppia (y, S) costituisce un'**OBIEZIONE** del giocatore i verso il giocatore j rispetto all'allocazione x se:

$$\begin{cases} i \in S \\ j \notin S \\ y_k > x_k \text{ per ogni } k \in S \end{cases}$$

In pratica i ha un'obiezione verso j se riesce a proporre una coalizione S che include i ma non j e una ripartizione di beni in cui ogni individuo di S guadagna di più rispetto all'allocazione x.

Direttamente per definizione, se i ha un'obiezione (y, S) contro j, anche tutti gli individui di S ce l'hanno.

Definizione 3.3.3. Data un'obiezione (y, S) del giocatore i verso il giocatore j rispetto all'allocazione x, una **CONTRO-OBIEZIONE** di j verso i è una coppia (z, T) dove T è una coalizione, z un vettore di $\mathbb{R}^T$ T-fattibile tali che:

$$\begin{cases} j \in T \\ i \notin T \\ z_k \geq x_k \text{ per ogni } k \in T \setminus S \\ z_k \geq y_k \text{ per ogni } k \in T \cap S \end{cases}$$

Quindi una contro-obiezione è un argomento ragionevole che j trova per minare l'autorevolezza della ripartizione proposta da i.

Definizione 3.3.4. Diremo che un'obiezione è **GIUSTIFICABILE** se non ne esistono contro-obiezioni.

Definizione 3.3.5. Il **BARGAINING SET** di un gioco $\mathcal{G} = (N, \nu)$ è l'insieme di tutte le allocazioni per le quali non vi è alcuna obiezione giustificata tra giocatori. Lo indichiamo $\mathcal{B}(\mathcal{G})$.

Esempio. Consideriamo il gioco cooperativo TU $\mathcal{G} = (N, \nu)$ in cui N=$\{a, b, c\}$, $\nu(\{a\}) = \nu(\{b\}) = \nu(\{c\}) = 0$, $\nu(\{a, b\}) = 20$, $\nu(\{a, c\}) = 30$, $\nu(\{b, c\}) = 40$ e $\nu(\{a, b, c\}) = 42$.
Focalizziamoci sull'allocazione $x = (x_a, x_b, x_c) = (8, 10, 24)$. Un'obiezione di c verso a rispetto ad x è ad esempio $((12, 28), \{b, c\})$, cui a può contro-obiettare con $((8, 12), \{a, b\})$.
Un'altra obiezione di c verso a può essere $((14, 26), \{b, c\})$. Tuttavia in questo caso a non può controbattere: infatti, la coalizione $\{a, b\}$ riceve 20 di pay-off e ciò non basta per organizzare una contro-obiezione, in quanto a ha bisogno di almeno 8 per sé e almeno 14 per b. Quindi $x \notin \mathcal{B}(\mathcal{G})$.
Vediamo ora l'allocazione $y = (y_a, y_b, y_c) = (4, 14, 24)$. Può a avere obiezioni giustificate verso c rispetto ad y? Prima di tutto notiamo che per avere

un'obiezione un'obiezione contro c, a deve formare la coalizione $\{a, b\}$. Sia $((\alpha_a, \alpha_b), \{a, b\})$ obiezione di a contro c, con

$$\begin{cases} \alpha_a + \alpha_b = 20 \\ \alpha_a \geq 4 \\ \alpha_b \geq 14 \end{cases} \tag{3.1}$$

Allora c può sempre contro-obiettare con $((\beta_b, \beta_c), \{b, c\})$ scegliendo ad esempio: $\begin{cases} \beta_b = \alpha_b \\ \beta_c = 40 - \beta_b \end{cases}$; per costruzione vale $\beta_b + \beta_c = 40$, $\beta_b \geq \alpha_b$ e se fosse $\beta_c < 24$:

$$40 - \beta_b < 24 \iff 40 - \alpha_b < 24 \iff \alpha_b > 16$$

e quest'ultima contrasta con la (3.1).
Si mostra in modo analogo che a non può avere obiezioni giustificate neanche contro b rispetto ad y e nemmeno b contro c. Quindi $y \in \mathcal{B}(\mathcal{G})$.

3.4 Complessità del Bargaining Set

In questa sezione mostriamo che il problema di decidere se un determinato vettore di pay-off sia nel Bargaining Set di un gioco a grafo sia Π_2^p-hard. Nello specifico useremo il corollario 1 del Capitolo 2, mostrando che il problema completo $NQSAT_2$ si riduce ad esso. Formuliamo più precisamente il problema:

Definizione 3.4.1. Il linguaggio **BARGAINING SET** (BS) consiste delle coppie $(\mathcal{G}, x)$ dove $\mathcal{G} = (N, \nu)$ è un gioco a grafo ed $x \in \mathcal{B}(\mathcal{G})$.

Ricordando come funziona una riduzione tra problemi, quello che faremo sarà fornire una trasformazione polinomiale che trasformi formule booleane in coppie (gioco a grafo, reso attraverso il suo grafo associato-allocazione) tale che la formula è $\forall\exists$-soddisfacibile se e solo se l'allocazione si trova nel Bargaining Set del gioco.

Usando le note proprietà logiche distributive, associative e di De Moivre è possibile dimostrare che lo studio delle CNF sia senza perdita di generalità equivalente a quello delle generiche formule booleane:

Lemma 6. Qualunque formula booleana può essere resa come CNF ed anche come 3CNF in modo equivalente, sfruttando una trasformazione polinomiale.

Inoltre è facile notare direttamente per definizione che se una formula φ è equivalente a una formula ψ allora anche $\forall\varphi$ e $\exists\varphi$ sono rispettivamente equi-

valenti a $\forall\psi$ e $\exists\psi$. In particolare non è restrittivo supporre che le istanze di $NQSAT_2$ siano formule

$$\varphi = \varphi(\boldsymbol{\alpha}, \boldsymbol{\beta})$$

tali che $\forall\boldsymbol{\alpha}\exists\boldsymbol{\beta}\varphi(\boldsymbol{\alpha}, \boldsymbol{\beta})$ è soddisfacibile, dove (si noti il grassetto) $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$, $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_r)$ e φ è congiunzione di m clausole contenenti ciascuna tre o meno letterali. Questa notazione ci accompagnerà per il resto del paragrafo.
Nel seguito supponiamo di aver fissato una codifica dei simboli delle variabili booleane $\boldsymbol{\alpha}$ e $\boldsymbol{\beta}$ e dei connettivi $\neg, \vee, \wedge$, così come le parentesi "(" e ")", in modo tale che ogni stringa binaria sia interpretabile come una sequenza di tali simboli e che sequenze che indicano "formule mal formate" siano individuabili facilmente.
Per procedere più speditamente utilizziamo uno stratagemma, vale a dire selezioniamo un sotto-linguaggio di $NQSAT_2$, dato da ***formule modificate*** $\varphi = \varphi(\boldsymbol{\alpha}, \boldsymbol{\beta})$, in cui:

1. il numero di variabili quantificate esistenzialmente è almeno superiore a quello di variabili quantificate universalmente (ossia $r \geq n$);

2. le prime $2n$ clausole abbiano la seguente forma, a meno di riordinarle:

$$(\alpha_1 \vee \neg\beta_1) \wedge (\neg\alpha_1 \vee \beta_1) \wedge \cdots \wedge (\alpha_n \vee \neg\beta_n) \wedge (\neg\alpha_n \vee \beta_n); \tag{3.2}$$

3. le ultime $m - 2n$ clausole, comprendano solo le variabili β_k (non le α_k) e ciascuna contenga al più 3 letterali;
   in particolare, per ogni $k \in \{1, \ldots, n\}$ la variabile α_k è contenuta in sole due clausole, che indicheremo rispettivamente $c_{i(k)}$ e $c_{\bar{i}(k)}$, ed assumeremo che compaia negata in $c_{\bar{i}(k)}$.

Indicheremo con $MNQSAT_2$ (M di "Modified") il linguaggio

$$MNQSAT_2 \coloneqq \{\varphi = \varphi(\boldsymbol{\alpha}, \boldsymbol{\beta}) \mid \varphi \text{ è una formula modificata}\}$$

Osservazione. Il linguaggio $MNQSAT_2$ è contenuto strettamente in $NQSAT_2$, poiché la struttura delle sue formule è alquanto specifica (ad esempio la formula $\varphi(\alpha_1, \alpha_2, \beta_1) = (\alpha_1 \vee \alpha_2 \vee \neg\beta_1)$ appartiene alla seconda classe ma non la prima). Si noti anche che la richiesta 2) delle formule modificate equivale alla richiesta che valga $\alpha_k \iff \beta_k$ per ogni k, in modo che la 3) sia una richiesta facilmente assolvibile, sostituendo i letterali α_k con i corrispondenti β_k.

È possibile dimostrare che il problema $MNQSAT_2$ ha la stessa complessità di $NQSAT_2$. In altre parole, le assunzioni che si fanno sulle formule modificate non rendono "più semplice" il problema, ma danno semplicemente una forma comune alle espressioni booleane, consentendoci di studiarle meglio:

Lemma 7. Il linguaggio $MNQSAT_2$ è Π_2^p-completo.

Vediamo nello specifico la riduzione da $MNQSAT_2$ a BARGAINING SET. La descrizione del grafo che segue è perlopiù discorsiva (e a mio parere meno oscura di una descrizione matematica): si tenga presente la **Figura 3.2** per avere un esempio tangibile della costruzione.
Per ogni formula $\varphi = \varphi(\boldsymbol{\alpha}, \boldsymbol{\beta})$, costruiamo il grafo pesato $BS(\varphi)$ in questo modo:

$$BS(\varphi) = ((N_{BS}, E_{BS}), w)$$

dove:

- l'insieme dei giocatori (vertici) N_{BS} è costituito da un vertice per ogni clausola c_j della formula, un vertice per ogni letterale $l_{i,j}$ della formula (al variare di $i \in \{1, 2, \ldots, r\}$ - numero della variabile - e $j \in \{1, \ldots, m\}$ - numero della clausola), e due ulteriori vertici (a e b);
- l'insieme degli archi E_{BS} include tre tipi di archi:
  1. **archi positivi**: un arco di peso 1 che collega il vertice-clausola c_j ad ogni vertice-letterale presente in c_j (al più tre archi di questo tipo per ogni $j \in \{1, \ldots, m\}$); un arco di peso 1 che collega b ad ogni nodo-letterale di tipo $\alpha_{i,j}$ o $\neg\alpha_{i,j}$ (cioè le variabili quantificate universalmente);
  2. **archi negativi**: un arco di peso $-m-1$ che collega ciascuna coppia di nodi-letterali distinti della stessa clausola c_j; un arco di peso $-m-1$ che collega b ad ogni nodo-letterale di tipo $\beta_{i,j}$ oppure $\neg\beta_{i,j}$ (cioè le variabili quantificate esistenzialmente); un arco di peso $-m-1$ che collega b ad ogni nodo-clausola c_j; un arco di peso $-m-1$ che collega nodi-letterali di clausole diverse in cui uno è il negato dell'altro;
  3. **arco di normalizzazione**: γ è l'arco che collega a e b di peso
  $$w(\gamma) = n + m - 1 - \sum_{e \in E_{BS} | e \neq \gamma} w(e)$$
- l'allocazione $\mathbf{x}$ che viene considerata è poi quella che assegna m ad a, $n-1$ a b e 0 ad ogni altro giocatore (si tratta di un'allocazione efficiente poiché per costruzione la somma dei pesi sugli archi del grafo è proprio $n+m-1$).

Si noti come la taglia della codifica di $BS(\varphi)$ e del peso dei suoi archi è polinomiale nel numero di variabili e di clausole di φ (cioè nella taglia di φ),

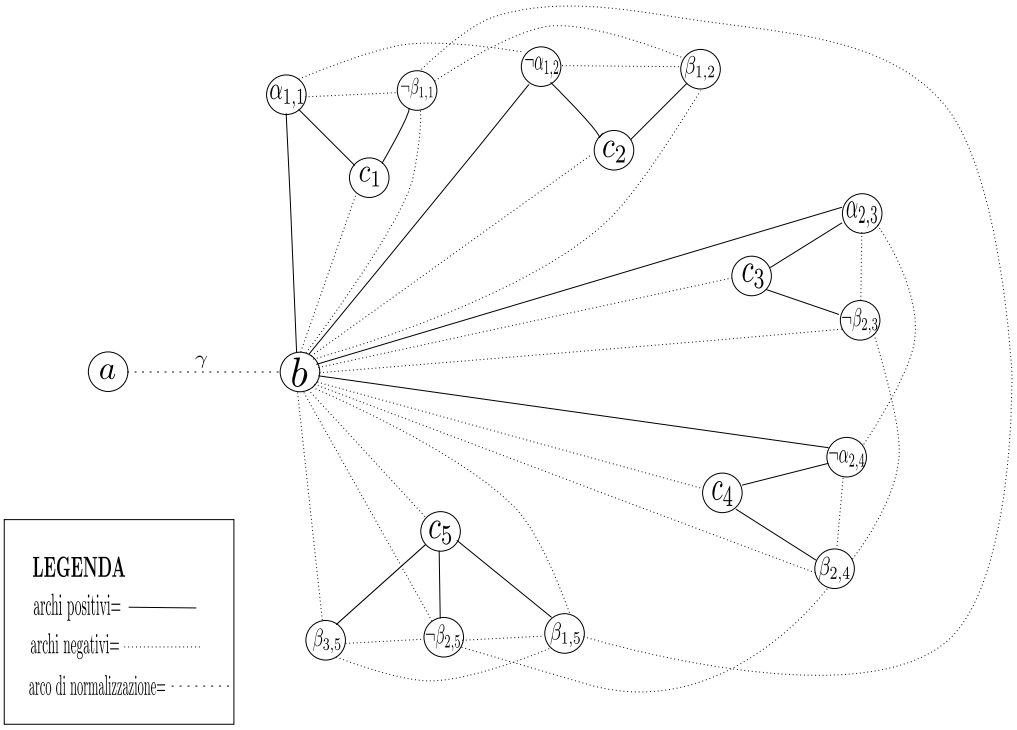

Figura 3.2: la costruzione di BS(φ) nel caso specifico $\varphi = (\alpha_1 \vee \neg\beta_1) \wedge (\neg\alpha_1 \vee \vee\beta_1) \wedge (\alpha_2 \vee \neg\beta_2) \wedge (\neg\alpha_2 \vee \beta_2) \wedge (\beta_1 \vee \neg\beta_2 \vee \beta_3)$

il che significa che è possibile costruire $BS(\varphi)$ da φ con un trasduttore polinomiale. Possiamo inoltre supporre che le formule booleane "mal formate" siano riconosciute e mandate in una non-istanza di BS.
In altre parole, la funzione tra stringhe $\varphi \mapsto BS(\varphi)$ è calcolabile in tempo polinomiale e sarà la nostra riduzione.

Il compito ora è quello di mostrare che se la formula φ è $\forall\exists$-soddisfacibile, la coppia $(BS(\varphi), \mathbf{x})$ è istanza di BARGAINING SET e viceversa.
Per farlo occorrono una serie di risultati preliminari tecnici.

Lemma 8. Sia $BS(\varphi) = ((N_{BS}, E_{BS}), w)$ il gioco a grafo associato alla formula φ, di arco di normalizzazione $\gamma = \{a, b\}$. Poniamo $D = \max_{\gamma \nsubseteq S \subseteq N_{BS}} \nu(S)$ la massima ricchezza ottenibile da coalizioni S di giocatori che non comprendono entrambi i giocatori a e b. Allora:

1. $m \geq 2n$;
2. $D \leq m$;
3. $D + w(e) < 0$ per ogni arco negativo $e \in E_{BS}$;

4. $w(\gamma) > 2m$.

Dimostrazione. 1) Dalle caratteristiche delle formule modificate.
2) Direttamente per costruzione, ricordando come si calcola il valore di una coalizione in un gioco a grafo. Si noti infatti come vi siano molti più archi negativi rispetto a quelli positivi[1] e che il peso dato ai primi è $-m-1$, molto più influente del peso 1.
3) Sia $e \in E_{BS}$ un arco negativo. Direttamente dalla 2) si ha:

$$D + w(e) \leq m - m - 1 = -1 < 0.$$

4) Dato che E_{BS} contiene più archi negativi che positivi, sicuramente la somma dei pesi degli archi di E_{BS} (escludendo γ) sarà minore o uguale del peso di un arco negativo:

$$\sum_{e \in E_{BS} | e \neq \gamma} w(e) \leq -m - 1$$

da cui anche

$$w(\gamma) = n - 1 + m - \sum_{e \in E_{BS} | e \neq \gamma} w(e) \geq n - 1 + m + m + 1 = n + 2m > 2m.$$

□

Proposizione 5. Sia $BS(\varphi) = ((N_{BS}, E_{BS}), w)$ il gioco a grafo associato alla formula φ e $\mathbf{x}$ l'allocazione già descritta. Valgono le seguenti:

1. Nessun giocatore ha alcuna obiezione giustificata rispetto ad $\mathbf{x}$ contro un giocatore-clausola o letterale.
2. Nessun giocatore ha alcuna obiezione giustificata rispetto ad $\mathbf{x}$ contro b.
3. Nessun giocatore tranne b ha alcuna obiezione giustificata rispetto ad $\mathbf{x}$ contro a.

Dimostrazione. 1) Vediamo il caso di obiezione contro un giocatore-clausola c_j: il caso di obiezione contro un giocatore-letterale è identico. Sia data un'obiezione contro c_j: dato che c_j in $\mathbf{x}$ riceve 0 ed è tale che $\nu(\{c_j\}) = 0$, egli potrà sempre contro-obiettare con $(\{c_j\}, 0)$.
2) Sia $p \in N_{BS}$ un giocatore che vuole obiettare contro b tramite la coalizione S.
Osserviamo che non si può avere $\nu(S) < 0$. Infatti, l'obiezione di p dovrebbe comprendere un vettore $y \in \mathbb{R}^S$ tale che $y_k > x_k$ per ogni $k \in S$: in particolare

[1] Un rapido conteggio mostra che il numero di archi positivi è al più $3m$ mentre il numero degli archi negativi è almeno $4m$.

anche $\nu(S) = y(S) > x(S)$, ma $x(S) \geq 0$ per costruzione di $\mathbf{x}$ e questo sarebbe assurdo.
Per il Lemma 8, S non può dunque contenere alcun arco negativo. Questo comporta che per ogni $k \in \{1, \ldots, n\}$ si abbia $|S \cap \{\alpha_{k,i(k)}, \neg\alpha_{k,\bar{i}(k)}\}| \leq 1$ (se infatti fosse $\{\alpha_{k,i(k)}, \neg\alpha_{k,\bar{i}(k)}\} \subseteq S$, in S dovrebbe apparire anche l'arco negativo che le collega).
Consideriamo allora la coalizione $T \subseteq \{b\} \cup \bigcup_{k=1}^{n}\{\alpha_{k,i(k)}, \neg\alpha_{k,\bar{i}(k)}\}$ tale che $|T| = n+1$, $T \cap S = \emptyset$ e $|T \cap \{\alpha_{k,i(k)}, \neg\alpha_{k,\bar{i}(k)}\}| = 1$ per ogni k: un tale T esiste per le considerazioni viste finora. Esso soddisfa anche che $\nu(T) = n$ e $x(T) = x_b = n-1$.
Si consideri poi il vettore $z \in \mathbb{R}^T$ tale che $z_b = x_b = n-1$ e $z_q = \frac{1}{n} > x_q = 0$ per ogni $q \in T,\ q \neq b$. Per costruzione, z è T-fattibile e (z, T) è una controobiezione all'obiezione di p verso b.
3) Supponiamo che un giocatore $p \neq b$ abbia un'obiezione (y, S) contro a rispetto ad $\mathbf{x}$. Dato che $a \notin S$, l'arco γ non sarà in S da cui per il Lemma 8 vale

$$\nu(S) \leq m. \tag{3.3}$$

Consideriamo la coalizione $T = \{a, b\}$ ed il vettore z che assegna m ad a e $w(\gamma) - m$ a b. Un tale z è chiaramente T-fattibile e si nota che (z, T) è controobiezione a quella sollevata da p. Infatti, $z_a = x_a$ e ci sono due casi:

- se $b \in S$ allora $z_b = w(\gamma) - m > m \geq y_b$ dove abbiamo usato il Lemma 8 nella disuguaglianza stretta e la (3.3) nell'altra;
- se $b \notin S$ allora $z_b = w(\gamma) - m > m \geq 2n > n-1 = x_b$ dove abbiamo usato due volte il Lemma 8.

In ogni caso possiamo concludere. □

Alla luce delle proprietà appena viste, possiamo limitare la nostra attenzione alle obiezioni di b contro a.
Sia dunque (y, S) un'obiezione di b contro a rispetto ad $\mathbf{x}$. Per definizione devono valere:

$$\begin{cases} y_b > x_b = n-1 \\ y_q > 0 = x_q \quad \forall q \in S,\ q \neq b \end{cases} \tag{3.4}$$

Inoltre, y è S-fattibile e quindi anche $y(S) = \nu(S) > n-1$. Sappiamo dal Lemma 8 che affinché valga $\nu(S) > n-1 \geq 0$, S non deve contenere alcun arco negativo. Ricordiamo anche che per definizione $b \in S$ ed $a \notin S$: data la costruzione di $BS(\varphi)$, l'unica possibilità che rimane è che S contenga esattamente un

giocatore per ogni variabile quantificata universalmente.

Quindi S dev'essere tale che: $\begin{cases} |S \cap \{\alpha_{k,i(k)}, \neg\alpha_{k,\bar{i}(k)}\}| = 1 \text{ per ogni } k \in \{1,\dots,n\} \\ |S| = n+1 \\ \nu(S) = n > n-1 \end{cases}$

dove le ultime due uguaglianze seguono dalla prima.

Questo ci permette di migliorare la (3.4) ottenendo:

$$\begin{cases} y(S) = n \\ y_b > x_b = n-1 \\ y_q > 0 = x_q \quad \forall q \in S,\ q \neq b \end{cases} \tag{3.5}$$

Osservazione. Si noti che le caratteristiche di S identificano uno e un solo assegnamento per le variabili $\boldsymbol{\alpha}$, dato da:

$\sigma_S(\alpha_k) = \begin{cases} 0 \text{ se } \alpha_{k,i(k)} \in S \\ 1 \text{ se } \alpha_{k,i(k)} \notin S \end{cases}$

Si noti come l'assegnamento σ_S sia "invertito" rispetto a come intuitivamente sarebbe più naturale definirlo: questo elemento sarà fondamentale in seguito. Viceversa, dato un assegnamento σ per le variabili α_k, esiste un unico S costituito da:

$$S = \{b\} \cup \bigcup_{k \in I_1} \{\alpha_{k,i(k)}\} \cup \bigcup_{k \in I_2} \{\neg\alpha_{k,\bar{i}(k)}\}$$

dove $I_1 = \{i \in \{1,\dots,n\} \mid \sigma(\alpha_i) = 0\}$ ed $I_2 = \{i \in \{1,\dots,n\} \mid \sigma(\alpha_i) = 1\}$. In altre parole, le obiezioni (y, S) di b contro a rispetto ad $\mathbf{x}$ devono rispettare la (3.5) e sono in corrispondenza biunivoca con i possibili assegnamenti di verità per le variabili quantificate universalmente.

Per concludere la dimostrazione della Π_2^p-hardness non rimane che dimostrare che φ è $\forall\exists$-soddisfacibile $\iff$ $\mathbf{x} \in \mathcal{B}(BS(\varphi))$.

Teorema 1. φ è $\forall\exists$-soddisfacibile $\iff$ $\mathbf{x} \in \mathcal{B}(BS(\varphi))$.

Dimostrazione. $\Rightarrow$) Sia (y, S) obiezione di b contro a rispetto ad $\mathbf{x}$ e mostriamo che non è giustificata. Sia σ_S assegnamento per le variabili $\boldsymbol{\alpha}$ associato ad S. Poniamo σ assegnamento che estende σ_S e soddisfa φ (esso esiste poiché per definizione di $\forall\exists$-soddisfacibilità, qualunque sia assegnamento per le α_k - compreso σ_S - esiste un assegnamento per le β_k che renda φ vera).

Basandoci su σ, consideriamo la coalizione T tale che valgano le seguenti:

$\begin{cases} T = \{a\} \cup \{c_1,\dots,c_m\} \cup \{l_{i,j} \mid l_{i,j} \text{ è letterale della clausola } c_j \text{ reso vero da } \sigma\} \\ T \cap S = \emptyset \\ |T| = 2m+1 \end{cases}$

Un tale T esiste poiché la terza condizione insieme alla prima sono facilmente assolvibili dal fatto che σ soddisfa φ (in particolare ogni clausola ha almeno un letterale vero secondo σ), mentre la seconda è rispettata grazie alla costruzione contro-intuitiva di σ_S. Si nota che $\nu(T) = m$ poiché una tale coalizione contiene solo m archi di peso 1 dati dagli archi clausola-letterale.
Si consideri poi il vettore $z \in \mathbb{R}^T$ tale che $z_a = x_a = m = \nu(T)$ e $z_q = x_q = 0$ per ogni $q \in T,\ q \neq a$. Per costruzione, (z, T) è contro-obiezione ad (y, S), da cui $\mathbf{x} \in \mathcal{B}(BS(\varphi))$.
$\Leftarrow$) Dimostriamo che se φ è non $\forall\exists$-soddisfacibile allora $\mathbf{x} \notin \mathcal{B}(BS(\varphi))$, cioè esiste un'obiezione giustificata di b contro a rispetto ad $\mathbf{x}$. Sia σ un assegnamento per le variabili $\boldsymbol{\alpha}$ tale che non sia possibile avere $\varphi(\boldsymbol{\alpha}, \boldsymbol{\beta})$ soddisfatta in alcun modo. Sia (y, S) un'obiezione di b contro a tale che $\sigma_S = \sigma$ e notiamo che essa è giustificata. Infatti, supponiamo per assurdo che ne esista una contro-obiezione (z, T): dato che $a \in T \setminus S$, deve valere $z_a \geq x_a = m$. Per il Lemma 8 si nota che ciò implica $z_a = m$. Infatti, proprio m è la massima ricchezza ottenibile da coalizioni che non comprendono sia a sia b e deve valere anche $z_q \geq 0$ per ogni $q \in T \setminus S,\ q \neq a$ e $z_q > y_q \geq 0$ per ogni $q \in T \cap S$ per definizione di contro-obiezione: non resta che la possibilità che sia $T \cap S = \emptyset$, $z_q = 0$ per ogni $q \in T,\ q \neq a$ e $z_a = m$.
In particolare quindi deve valere $\nu(T) = \sum_{i \in T} z_i = m$, il che caratterizza T in maniera molto precisa. Infatti, per il Lemma 8, T non può contenere alcun arco negativo, non contiene b e deve avere m archi positivi: l'unica possibilità è che

$$T = \{a\} \cup \{c_1, \ldots, c_m\} \cup \bigcup_{(i,j) \in I} \{l_{i,j}\}$$

dove i letterali $\{l_{i,j}\}$ che T contiene sono esattamente uno per ogni clausola e non si può verificare che letterali dati dalla negazione l'uno dell'altro siano entrambi in T (altrimenti in T dovrebbe ricadere anche l'arco negativo che li unisce). Ciò comporta che T codifichi un assegnamento σ_T per le variabili $\boldsymbol{\alpha}$ e $\boldsymbol{\beta}$ dato da:

$$\sigma_T(\alpha_k) = \begin{cases} 1 \text{ se } \alpha_{k,i(k)} \in T \\ 0 \text{ se } \neg\alpha_{k,\bar{i}(k)} \in T \end{cases} \quad ; \quad \sigma_T(\beta_k) = \begin{cases} 1 \text{ se } \exists j \in \{1, \ldots, m\} | \beta_{k,j} \in T \\ 0 \text{ se } \exists j \in \{1, \ldots, m\} | \neg\beta_{k,j} \in T \end{cases}$$

(mentre σ_T per le variabili β_k eventualmente non specificate può essere definito arbitrariamente).
Si nota che σ_T soddisfa $\varphi(\boldsymbol{\alpha}, \boldsymbol{\beta})$ poiché vi è un letterale vero secondo esso in ogni clausola c_j di φ. Inoltre, se indichiamo con $\sigma_T^{\boldsymbol{\alpha}}$ la restrizione di σ_T alle variabili $\boldsymbol{\alpha}$, dato che si ha $T \cap S = \emptyset$, vale anche $\sigma_T^{\boldsymbol{\alpha}} = \sigma_S$ per la costruzione controintuitiva

di σ_S. Tuttavia questo è assurdo in quanto possiamo ricavare $\sigma = \sigma_T^{\boldsymbol{\alpha}}$ e proprio σ era l'assegnamento che non poteva in alcun modo $\forall\exists$-soddisfare φ. Quindi la contro-obiezione (z, T) non può esistere e (y, S) è obiezione giustificata di b contro a rispetto a $\mathbf{x}$. □

Corollario 6. Il problema BARGAINING SET è Π_2^p-hard per giochi a grafo.

Osservazione. Si può poi dimostrare che il problema BARGAINING SET per giochi a grafo è in Π_2^p, da cui segue immediatamente la sua Π_2^p-completezza.

3.5 Il Kernel

In questa sezione introduciamo un altro famoso concetto di soluzione per giochi cooperativi TU: il ***kernel***.

Definizione 3.5.1. Dato un gioco $\mathcal{G} = (N, \nu)$ e dati due giocatori distinti i e j in N, denotiamo con $\boldsymbol{\mathcal{I}_{i,j}}$ l'insieme di tutte le possibili coalizioni che contengono l'individuo i ma non j.

Definizione 3.5.2. Dato un gioco $\mathcal{G} = (N, \nu)$, una coalizione S ed un'allocazione $x \in X(\mathcal{G})$, chiamiamo **ECCESSO** di S rispetto ad x il valore reale

$$e(S, x) \coloneqq \nu(S) - x(S)$$

Chiamiamo poi **SURPLUS** del giocatore i sul giocatore j rispetto all'allocazione x la quantità

$$s_{i,j}(x) \coloneqq \max_{S \in \mathcal{I}_{i,j}} e(S, x) = \max_{S \in \mathcal{I}_{i,j}} (\nu(S) - x(S))$$

Intuitivamente esso rappresenta il più alto pay-off che il giocatore i può guadgnare (o il minimo che egli può perdere, nel caso si parli di valori negativi) senza la cooperazione di j. In particolare, i ha più "potere contrattuale" di j se $s_{i,j}(x) > s_{j,i}(x)$.

Chiaramente, j può non sentirsi minacciato da questa situazione se ad esempio si ha che $\nu(\{j\}) = x_j$ in quanto in tal caso è in grado di ottenere lo stesso pay-off lavorando "senza coalizioni", in autonomia.

Definizione 3.5.3. Diciamo che il giocatore i **HA PIÙ PESO** del giocatore j se: $\begin{cases} s_{i,j}(x) > s_{j,i}(x) \\ x_j > \nu(\{j\}) \end{cases}$

Il kernel di un gioco è proprio l'insieme di tutte le allocazioni in cui nessun giocatore ha più peso degli altri:

Definizione 3.5.4. Il **KERNEL** di un gioco $\mathcal{G} = (N, \nu)$ cooperativo TU è dato da:

$$\mathcal{K}(\mathcal{G}) \coloneqq \{x \in X(\mathcal{G}) | s_{i,j}(x) > s_{j,i}(x) \Rightarrow x_j = \nu(\{j\}),\ \forall i, j \in N, i \neq j\}$$

Esempio. Riprendiamo l'esempio del gioco cooperativo TU $\mathcal{G} = (N, \nu)$ in cui N=$\{a, b, c\}$, $\nu(\{a\}) = \nu(\{b\}) = \nu(\{c\}) = 0$, $\nu(\{a, b\}) = 20$, $\nu(\{a, c\}) = 30$, $\nu(\{b, c\}) = 40$ e $\nu(\{a, b, c\}) = 42$.
L'allocazione $x_a = 4$, $x_b = 14$ e $x_c = 24$, che era nel Bargaining Set del gioco, si trova anche nel suo kernel. Infatti essa è efficiente, individualmente razionale ed ogni giocatore riceve un pay-off strettamente maggiore di quello che otterrebbe agendo da solo. Pertanto, affinchè $x \in \mathcal{K}(\mathcal{G})$ non deve verificarsi mai che $s_{i,j}(x) > s_{j,i}(x)$, il che si dà in quanto $s_{i,j}(x) = 2\ \ \forall i, j \in N, i \neq j$.

3.6 Complessità del Kernel

Vediamo ora com'è possibile che il problema di stabilire se una certa allocazione si trovi o meno nel kernel di un gioco a grafo sia Δ_2^p-completo.
In questa sezione, a meno che non diversamente specificato, ci concentreremo su formule booleane soddisfacibili: i linguaggi ammessi saranno contenuti nella grande famiglia delle formule booleane soddisfacibili che è SAT.

Definizione 3.6.1. Indichiamo con **IN-KERNEL** il seguente problema:

$$\text{IN-KERNEL} = \{(\mathcal{G}, x) |\ \mathcal{G} \text{ è un gioco a grafo ed } x \in \mathcal{K}(\mathcal{G})\}$$

Per mostrare la Δ_2^p-hardness sfrutteremo una riduzione polinomiale da un celebre problema Δ_2^p-completo, che ora introduciamo.

Osservazione. Data una formula 3CNF $\varphi = \varphi(\boldsymbol{\alpha})$ con $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ vettore di variabili booleane ordinate secondo gli indici crescenti, e dato un assegnamento σ per le variabili $\boldsymbol{\alpha}$, è naturale associare a σ un vettore n-dimensionale la cui coordinata i-esima è 0 (rispettivamente 1) se e solo se $\sigma(\alpha_i) = 0$ (rispettivamente 1). Interpretando tali vettori come stringhe binarie, è possibile costruire un ordinamento tra tutti i possibili assegnamenti per le variabili, al cui vertice vi è l'assegnamento massimo (1,1,...,1) e alla cui base quello minimo, (0,0,...,0), che dà falso a tutte le variabili.

Definizione 3.6.2. Data una formula 3CNF $\varphi = \varphi(\boldsymbol{\alpha})$ con $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_n)$ vettore di variabili booleane ordinate secondo gli indici crescenti, il problema

FIRSTMAX-SAT è dato dallo stabilire se l'assegnamento massimo che soddisfa φ assegna vero ad α_1, la prima variabile dell'ordine - la meno significativa[2]:

$$\text{FIRSTMAX-SAT}=\{\varphi=\varphi(\boldsymbol{\alpha})|\ \varphi \text{ è una 3CNF e l'assegnamento massimo che soddisfa } \varphi \text{ assegna vero ad } \alpha_1\}.$$

Non è restrittivo supporre che qualunque formula di FIRSTMAX-SAT contenga almeno una clausola con almeno due letterali (in caso contrario si possono inserire alcune variabili ausiliarie sfruttando le equivalenze della logica booleana).

Lemma 9. FIRSTMAX-SAT è un problema Δ_2^p-completo.

Nelle prossime righe vediamo come costruire una riduzione polinomiale da FIRSTMAX-SAT a IN-KERNEL: l'obiettivo ancora una volta è quello di trasformare una formula 3CNF $\varphi=\varphi(\boldsymbol{\alpha})$ con $\boldsymbol{\alpha}=(\alpha_1,\ldots,\alpha_n)$ con $\varphi=c_1\wedge\cdots\wedge c_m$ in un grafo pesato $K(\varphi)=((N_K,E_K),w)$ tale che:

- l'insieme dei vertici N_K è dato da un giocatore-variabile α_i per ogni $i\in\{1,\ldots,n\}$, un giocatore-clausola c_j per ogni $j\in\{1,\ldots,m\}$ ed un giocatore letterale $l_{i,j}$ (con $l_{i,j}=\alpha_{i,j}$ o $l_{i,j}=\neg\alpha_{i,j}$) per ogni letterale l_i in c_j; inoltre vi sono due giocatori ulteriori, a e b;

- l'insieme degli archi E_K consiste di archi di tre tipi:

  1. **archi positivi**: un arco di peso 2^{n+3} che collega ogni nodo-clausola c_j ad ogni nodo-letterale $l_{i,j}$ che compare in c_j; un arco che collega b al nodo-variabile α_i di peso 2^i per ogni $i\in\{1,\ldots,n\}$; un arco che collega a al nodo-variabile α_i di peso 2^i per ogni $i\in\{2,\ldots,n\}$; un arco che collega a alla variabile α_1 di peso 2^1+2^0, ossia 3;

  2. **archi negativi**: un arco di peso -2^{n+m+7} che collega ogni coppia di nodi-letterali distinti della stessa clausola; un arco di peso -2^{n+m+7} che collega nodi-letterali che in clausole diverse sono uno il negato dell'altro; un arco di peso -2^{n+m+7} che collega il nodo-variabile α_i al nodo-letterale $\neg\alpha_{i,j}$ per ogni variabile α_i che compare negata nella clausola c_j;

[2]Nella letteratura spesso lo stesso linguaggio è definito senza la richiesta di avere formule soddisfacibili: la nostra variazione, che ne identifica un sotto-linguaggio, rimane della stessa complessità Δ_2^p. Dato che ci siamo ristretti al panorama delle formule soddisfacibili, il complementare del nostro linguaggio FIRSTMAX-SAT sarà costituito da formule soddisfacibili per cui l'assegnamento massimo soddisfacente non assegna vero ad α_1.

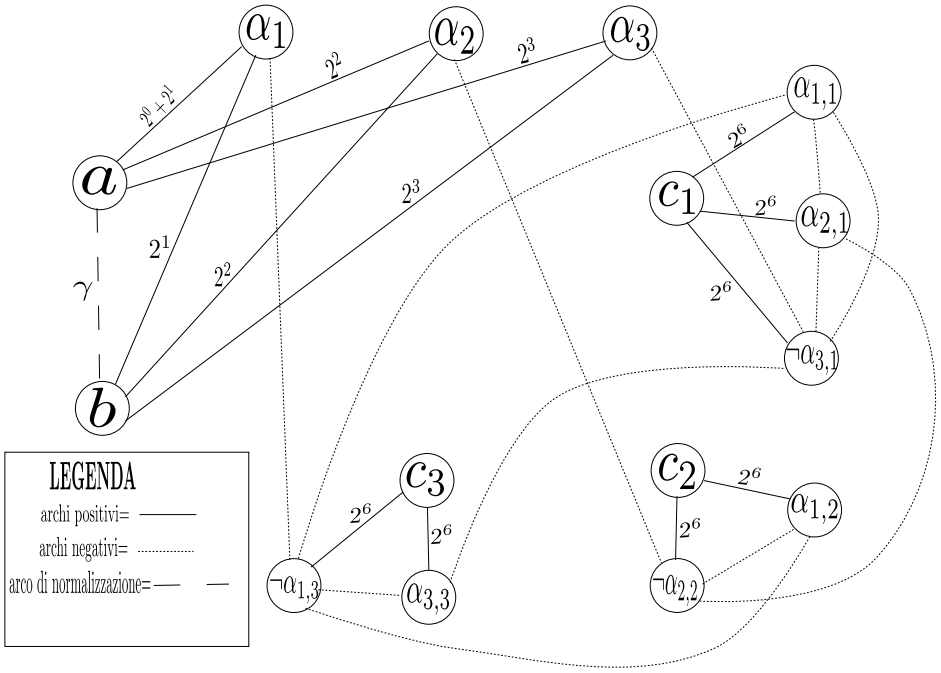

Figura 3.3: la costruzione di $K(\varphi)$ nel caso specifico di $\varphi = (\alpha_1 \vee \alpha_2 \vee \neg\alpha_3) \wedge \wedge(\alpha_1 \vee \neg\alpha_2) \wedge (\neg\alpha_1 \vee \alpha_3)$.

3. **arco di normalizzazione**: γ è l'arco che collega a e b di peso

$$w(\gamma) = 1 - \sum_{e \in E_K | e \neq \gamma} w(e)$$

- l'allocazione x che si considera è poi quella che assegna 1 ad a e 0 ad ogni altro giocatore (si noti che risulta efficiente poiché la somma dei pesi di tutti gli archi è proprio 1 per costruzione).

Si osserva che la taglia di $K(\varphi)$ è polinomiale nel numero di clausole e variabili che costituiscono φ (cioè nella taglia di φ). Inoltre, anche la taglia dei pesi sugli archi di $K(\varphi)$ è polinomiale in n ed m. In altre parole, la funzione che associa $\varphi \mapsto K(\varphi)$ è calcolabile da un trasduttore in tempo polinomiale.
Un buon modo per vedere la costruzione esplicita è la **Figura 3.3**.

Quello che dobbiamo mostrare è che l'assegnamento massimo che soddisfi φ assegna vero ad α_1 se e solo se $x \in \mathcal{K}(\mathcal{G})$. Ancora una volta per farlo sfruttiamo la costruzione e una serie di risultati preliminari.

Lemma 10. Sia $K(\varphi) = ((N_K, E_K), w)$ il gioco a grafo associato alla 3CNF φ, di arco di normalizzazione γ ed x l'allocazione già presentata. Poniamo

$D = \max_{\gamma \not\subseteq S \subseteq N_K} \nu(S)$ la massima ricchezza ottenibile da coalizioni S di giocatori che non comprendono entrambi i giocatori a e b. Allora:

1. $w(\gamma) \geq D + 1$;
2. $D + w(e) < 0$, per ogni arco negativo $e \in E_K$;
3. per ogni giocatore $i \notin \{a, b\}$, si ha $\max_{S \in \mathcal{I}_{i,a}} e(S, x) \leq \max_{S \in \mathcal{I}_{a,i}} e(S, x)$.

Dimostrazione. 1) Poniamo P la somma dei pesi di tutti gli archi positivi in E_K. Un semplice conteggio permette di avere la seguente:

$$P \leq 3 \cdot m \cdot 2^{n+3} + 2 \cdot \sum_{i=1}^{n} 2^i + 2^0 \leq 2^{n+m+5} + \sum_{i=0}^{n} 2^{i+1} \leq 2^{n+m+5} + 2^{n+2} \leq 2^{n+m+6}$$

dove nella seconda maggiorazione abbiamo usato che $3m \leq 2^{m+2}$, la quale è verificabile graficamente per ogni $m \in \mathbb{R}$. In particolare si ha $2P \leq 2^{n+m+7}$. Dato che φ contiene almeno un arco negativo (φ ha almeno una clausola con due letterali), vale che $w(\gamma) = 1 - \sum_{e \in E_K | e \neq \gamma} w(e) \geq 1 - P + 2^{n+m+7} \geq 1 - P + 2P = 1 + P \geq D + 1$, il che prova la 1).
2) Segue dal punto precedente, in quanto per ogni arco negativo $e \in E_K$ si ha: $w(e) = -2^{n+m+7} \leq -2P < -D$, dove la disuguaglianza stretta è frutto del fatto che sia P sia D sono quantità strettamente positive tali che $P \geq D$.
3) Siano $i \notin \{a, b\}$ e $S \in \mathcal{I}_{i,a}$. Si consideri la coalizione $T = \{a, b\} \in \mathcal{I}_{a,i}$. Si ha: $e(T, x) = \nu(T) - x(T) = \nu(\{a, b\}) - 1 = w(\gamma) - 1 \geq D$ per la prima parte ed anche $e(S, x) = \nu(S) - x(S) = \nu(S) \leq D$ per definizione, da cui vale a maggior ragione la stima richiesta. □

Dato il grafo $K(\varphi) = ((N_K, E_K), w)$ associato alla formula φ, si nota che a è l'unico giocatore che riceve nell'allocazione x una ricchezza superiore al valore di pay-off della propria coalizione-singoletto. La definizione di Kernel garantisce dunque che $x \in \mathcal{K}(\mathcal{G})$ se e solo se per ogni giocatore $i \in N, i \neq a$, valga $s_{i,a}(x) \leq s_{a,i}(x)$; ossia sostituendo le definizioni:

$$x \in \mathcal{K}(\mathcal{G}) \iff \max_{S \in \mathcal{I}_{i,a}} e(S, x) \leq \max_{S \in \mathcal{I}_{a,i}} e(S, x).$$

Il Lemma 10 permette di restringere ulteriormente il campo:

$$x \in \mathcal{K}(\mathcal{G}) \iff \max_{S \in \mathcal{I}_{b,a}} e(S, x) \leq \max_{S \in \mathcal{I}_{a,b}} e(S, x) \tag{3.6}$$

Quello che facciamo ora è caratterizzare entrambi i membri della disuguaglianza nella (3.6) attraverso assegnamenti che soddisfano φ.
A tal fine, per ogni assegnamento σ per le variabili α_k, denotiamo con $\sigma \models \varphi$ il fatto che σ soddisfi φ.

Proposizione 6. Nelle stesse notazioni si hanno le seguenti:

1. $\max_{S\in\mathcal{I}_{b,a}} e(S,x) = m\cdot 2^{n+3} + \max_{\sigma\models\varphi}\sum_{\sigma(\alpha_i)=1} 2^i$;
2. $\max_{S\in\mathcal{I}_{a,b}} e(S,x) = m\cdot 2^{n+3} + \max_{\sigma\models\varphi}\left(|\{\alpha_1|\sigma(\alpha_1)=1\}| + \sum_{\sigma(\alpha_i)=1} 2^i\right) - 1$

Dimostrazione. 1) Prima di tutto notiamo che per definizione il valore $\max_{S\in\mathcal{I}_{b,a}} e(S,x)$ è esattamente $\max_{S\in\mathcal{I}_{b,a}} \nu(S)$. Indichiamo con S_* una coalizione di $\mathcal{I}_{b,a}$ che ottiene massimo pay-off. Dato che $\nu(\{b\}) = 0$ e $\{b\} \in \mathcal{I}_{b,a}$, sicuramente deve valere anche $\nu(S_*) \geq 0$: per il Lemma 10, S_* non deve contenere alcun arco negativo. Possiamo dunque porre $C = \{$giocatori-clausola c_j | esattamente un letterale $l_{i,j}$ si trova in $S_*\}$, sfruttando il fatto che non ci possono essere più letterali della stessa clausola in S_* (vi sarebbe infatti anche l'arco che li collega in tal caso). Dato che non ci sono archi negativi che hanno estremi in nodi clausola, possiamo supporre che tutti i nodi-clausola c_j si trovino in S_* (tra di essi, quelli di C avranno un nodo-letterale in S_*, quelli fuori da C invece no). Abbiamo che:
$\nu(S_*) = |C|\cdot 2^{n+3} + \sum_{\alpha_i\in S_*} 2^i$ per costruzione di $K(\varphi)$. Poniamo $\hat{\sigma}$ il seguente assegnamento di verità:

$$\hat{\sigma}(\alpha_i) = \begin{cases} 1 \text{ se } \exists j \in \{1,\dots,m\}|\ \alpha_{i,j} \in S_* \\ 0 \text{ se } \exists j \in \{1,\dots,m\}|\ \neg\alpha_{i,j} \in S_* \end{cases}$$

Dato che S_* non ha archi negativi, $\hat{\sigma}$ è sicuramente coerente e soddisfa tutte le clausole di φ i cui giocatori si trovano in C. Si tenga presente che $\hat{\sigma}$ può trattarsi di un assegnamento parziale. Si osserva che se un vertice-letterale $\neg\alpha_{i,j} \in S_*$, allora necessariamente $\alpha_i \notin S_*$ per non avere archi negativi in S_*; viceversa se $\alpha_{i,j} \in S_*$ (e quindi $\neg\alpha_{i,j} \notin S_*$) dovrà essere $\alpha_i \in S_*$ dato che S_* realizza la massima ricchezza. Possiamo dunque considerare un nuovo assegnamento, σ_{S_*}:

$$\sigma_{S_*}(\alpha_i) = \begin{cases} 1 \text{ se } \alpha_i \in S_* \\ 0 \text{ se } \alpha_i \notin S_* \end{cases}$$

ed esso sicuramente coincide con $\hat{\sigma}$ nelle variabili in cui quest'ultimo è definito. Si ricorda ora che φ è soddisfacibile e in tal caso è facile vedere che $|C| = m$, da cui $\nu(S_*) = m\cdot 2^{n+3} + \sum_{\alpha_i\in S_*} 2^i$. Allora, σ_{S_*} è assegnamento che soddisfa φ e quindi $\nu(S_*) = m\cdot 2^{n+3} + \sum_{\alpha_i|\sigma_{S_*}(\alpha_i)=1} 2^i \leq m\cdot 2^{n+3} + \max_{\sigma\models\varphi}\sum_{\sigma(\alpha_i)=1} 2^i$. Concludiamo la dimostrazione mostrando che tale disuguaglianza non può essere stretta. Supponiamo per assurdo che esista un assegnamento $\bar{\sigma}$ che soddisfi φ e tale che $\nu(S_*) < m\cdot 2^{n+3} + \sum_{\alpha_i|\bar{\sigma}_{S_*}(\alpha_i)=1} 2^i$. Basandoci su $\bar{\sigma}$ possiamo costruire la coalizione $\bar{S}$ tale che:

- $\{b\} \cup \{c_1,\dots,c_m\} \subseteq \bar{S}$;

- $\alpha_i \in \bar{S}$ per ogni variabile α_i tale che $\bar{\sigma}(\alpha_i) = 1$;
- esattamente un letterale $l_{i,j}$ che è reso vero da $\bar{\sigma}$ si trova in $\bar{S}$ per ogni clausola c_j;
- nessun altro giocatore si trova in $\bar{S}$.

Tale coalizione avrebbe valore superiore a quello di S_* per costruzione, il che sarebbe assurdo.
2) Si procede come al punto precedente: le poche differenze sono nel fatto che per ogni $S \in \mathcal{I}_{a,b}$ si ha $x(S) = 1$ (non più 0), ed i pesi associati agli archi, i quali dipendono dal valore di verità che si dà ad α_1, il che giustifica la presenza dell'addendo $|\{\alpha_1|\sigma(\alpha_1) = 1\}|$ in funzione dell'assegnamento σ. □

Possiamo riscrivere alla luce di ciò la disuguaglianza nella (3.6) come:

$$1 + \max_{\sigma \models \varphi} \sum_{\alpha_i|\sigma(\alpha_i)=1} 2^i \leq \max_{\sigma \models \varphi} \left(\sum_{\alpha_i|\sigma(\alpha_i)=1} 2^i + (|\{\alpha_1|\sigma(\alpha_1) = 1\} \right)$$

e quindi concludere che:

$x \in \mathcal{K}(\mathcal{G}) \iff \alpha_1$ è vera nell'assegnamento massimo che soddisfa φ, da cui:

Corollario 7. Il problema IN-KERNEL è Δ_2^p-hard.

Passiamo ora a dimostrare che IN-KERNEL è un problema in Δ_2^p, cioè che ne esiste un solutore deterministico polinomiale che sfrutta un oracolo in NP.

Proposizione 7. In un gioco a grafo $\mathcal{G} = (N, \nu)$, il calcolo del valore associato a una coalizione $S \subseteq N$ è polinomiale nella taglia del gioco stesso.

Dimostrazione. Ricordiamo che il modo più comune per codificare in binario un grafo pesato è quello di listare ordinatamente tutti i vertici (eventualmente rinominandoli) e successivamente elencare coppie di vertici (per indicare l'arco che collega tali vertici) ed infine listare una serie di stringhe binarie (che costituiscono, ordinatamente, il peso degli archi del grafo).
Dopo questa premessa, il risultato è semplice. Sia $S \subseteq N$ una coalizione. Consideriamo il seguente algoritmo, che agisce direttamente sulla stringa che codifica il grafo:

- si eliminano tutti i vertici;

- si eliminano tutti gli archi (compresi del loro peso) che hanno almeno uno dei vertici fuori da S;
- si sommano i pesi rimanenti.

Dato che tutte le operazioni descritte sono polinomiali nella taglia dell'input, l'algoritmo proposto sarà polinomiale e permette di calcolare il peso della coalizione S. In altre parole, il calcolo $\nu(S)$ è in FP. □

Teorema 2. Il problema IN-KERNEL è Δ_2^p-completo.

Dimostrazione. Rimane solo da mostrare l'appartenza del problema alla classe Δ_2^p. Per prima cosa osserviamo che dato un gioco a grafo $\mathcal{G} = (N, \nu)$, le operazioni che permettono di controllare se un vettore $x \in \mathbb{R}^N$ sia un'allocazione per $\mathcal{G}$ sono polinomiali deterministiche per la Proposizione 7 in quanto esse sono date da:

1. calcolare il valore $\nu(\{i\})$ per ogni $i \in N$;
2. controllare che sia $\nu(\{i\}) \leq x_i$ per ogni $i \in N$;
3. calcolare che valga $\sum_{i \in N} x_i = \nu(N)$.

Dopo aver stabilito che $x \in X(\mathcal{G})$, occorre controllare che per ogni coppia di giocatori distinti i e j tali che $\nu(\{j\}) < x_j$, valga $s_{i,j}(x) \leq s_{j,i}(x)$. Per il calcolo di $s_{i,j}(x)$ possiamo procedere come segue:

- si pone A la somma di tutti i pesi positivi sugli archi del grafo (0 in caso non ce ne siano) e B la somma di tutti i pesi negativi sugli archi del grafo (0 in caso non ce ne siano);
- si effettua una ricerca binaria di estremi A e B, la quale viene coadiuvata da un oracolo che, per ogni valore di dimezzamento h, decide istantaneamente se esiste o meno un $S \in \mathcal{I}_{i,j}$ tale che $e(S, x) > h$.

Per comodità poniamo n la taglia dell'input del problema. Operativamente, la ricerca binaria potrebbe essere impostata così:
while $A - B > \varepsilon$:
    $mid = \frac{A+B}{2}$
    if $\exists S \in \mathcal{I}_{i,j} \mid e(S, x) > mid$: (oracolo lo decide in uno step)
        $B = mid$
    else :
        $A = mid$
dove il parametro $\varepsilon \in \mathbb{R}^+$ rappresenta la precisione.

Al termine del ciclo, i termini A e B, che sono rappresentati da taglia polinomiale in n, sono distanti meno di ε: scegliendo accuratamente tale parametro (ad esempio $\varepsilon = 2^{-p(n)}$ per un certo polinomio $p(n)$), il numero di chiamate sarà $\mathcal{O}(log(2^{p(n)}) = \mathcal{O}(p(n))$, cioè polinomiale nella taglia dell'input del problema[3]. Il costo di ciascuna chiamata è polinomiale poiché essa consiste di una somma, una divisione e una verifica istantanea, quindi in tempo polinomiale un ciclo simile individua il massimo eccesso $e(S, x)$ con $S \in \mathcal{I}_{i,j}$. In modo analogo è calcolabile in FP con aiuto dell'oracolo anche $s_{j,i}(x)$ ed il loro confronto sarà polinomiale nella loro taglia combinata (ancora polinomiale in n).
Pertanto la prova è conclusa a meno di dimostrare che un oracolo così descritto esista in NP. Tuttavia ciò è facile perché basta considerare la macchina non deterministica che indovina sul nastro possibili coalizioni S ed in secondo tempo controlla che esse siano in $\mathcal{I}_{i,j}$ e che $e(S, x) > h$. Una tale routine esterna è chiaramente non deterministica e polinomiale (fissato un S, il calcolo di $e(S, x)$ è polinomiale per la Proposizione 7, così come è polinomiale il controllo che $i \in S$ e che $j \notin S$). Quindi complessivamente il calcolo di $s_{i,j}(x)$ è in FP^{NP} e decidere se un vettore $x \in X(\mathcal{G})$ o meno è possibile farlo in tempo polinomiale con un oracolo in NP, il che conclude. □

[3]Si ricorda che una ricerca binaria "classica" su intervallo reale $[a, b]$ non individua un valore esatto, ma un intervallo di incertezza con precisione ε. Dopo k passi l'intervallo di ricerca si dimezza k volte, quindi il numero di iterazioni necessario per raggiungere precisione ε è almeno $log_2(\frac{|b-a|}{\varepsilon})$. Nel nostro contesto tuttavia i numeri in questione hanno rappresentazione polinomiale in n e questo, insieme ad un'oculata scelta della precisione ε, ci permette di avere la garanzia che la ricerca abbia un numero di chiamate polinomiale in n.

Conclusioni

Il presente elaborato è nato con l'idea di essere una panoramica quanto più breve ed autocontenuta possibile sulla gerarchia polinomiale, le sue proprietà ed i rapporti con le altre classi di complessità. A questo proposito si presti attenzione alle Proposizioni 1, 2 e 3 e ai Corollari seguenti, così come alla presentazione dei problemi completi per gerarchia polinomiale più comuni delle pagine successive.

Durante la stesura, è stato piuttosto naturale mostrare un'applicazione di alcuni dei concetti sviluppati nella prima parte ad un campo apparentemente molto lontano, la teoria dei giochi. L'idea di aggiungere una parte corposa di esempi di problemi di natura diversa dalla logica booleana, tratti dalla teoria dei giochi cooperativi, ha permesso di definire ed esemplificare i concetti del Bargaining Set e del Kernel di un gioco a grafo.
Allo stesso tempo ha dato l'occasione di trattare brevemente dei giochi compatti e di introdurre il problema della loro rappresentazione concisa, un campo molto prolifico. I risultati di Π_2^p-hardness (Corollario 6) e di Δ_2^p-completezza (Teorema 2) dei problemi del Bargaining Set e del Kernel rispettivamente sono piuttosto recenti (2011) ed aprono il campo a numerose generalizzazioni, ad esempio in termini di rappresentazioni di giochi dello stesso tipo secondo altri metodi (diversi dai grafi pesati); inoltre tali risultati possono essere studiati aggiungendo (o indebolendo) ipotesi sulla modellizzazione dei giochi stessi (giochi ad utilità non trasferibile, oppure giochi ad albero, per dirne alcuni).

Per ultimo non vorrei mancare di osservare che le dimostrazioni sui grafi sono a mio parere molto fantasiose ed acute e danno appena un assaggio di quanto possa essere complesso e variegato il mondo delle riduzioni dell'Informatica Teorica.

Sperando di fare un favore al lettore interessato, pongo di seguito una bibliografia con alcuni validissimi testi ed articoli di ricerca per completare ed approfondire gli argomenti presentati.

Bibliografia

[1] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[2] Robert J Aumann and Michael Maschler. *The bargaining set for cooperative games*. Princeton University Princeton, NJ, 1961.

[3] José L Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural complexity*, volume 1. Cambridge University Press, 1994.

[4] Morton Davis and Michael Maschler. The kernel of a cooperative game. *Naval Research Logistics Quarterly*, 1965.

[5] Xiaotie Deng and Christos H Papadimitriou. On the complexity of cooperative solution concepts. *Mathematics of operations research*, 1994.

[6] Gianluigi Greco, Enrico Malizia, Luigi Palopoli, and Francesco Scarcello. On the complexity of compact coalitional games. In *IJCAI*, 2009.

[7] Gianluigi Greco, Enrico Malizia, Luigi Palopoli, and Francesco Scarcello. On the complexity of core, kernel, and bargaining set. *Artificial Intelligence*, 2011.

[8] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. *Introduction to automata theory, languages, and computation*. ACM New York, NY, USA, 2001.

[9] Martin J Osborne et al. *An introduction to game theory*. Springer, 2004.

[10] Martin J Osborne and Ariel Rubinstein. *A course in game theory*. MIT press, 1994.

[11] Michael Sipser. *Introduction to the Theory of Computation*. ACM New York, NY, USA, 1996.

Ringraziamenti

Prima di tutto vorrei ringraziare il mio relatore, il Professor **Enrico Malizia** che si è dimostrato gentilissimo e sempre assai disponibile per guidarmi e rispodere alle mie domande: senza di lui sicuramente questo lavoro non sarebbe tra le vostre mani.
In secondo luogo vorrei ringraziare la mia famiglia, che mi ha sempre supportato ed è stata validissimo appoggio nei momenti di difficoltà che ho affrontato nel percorso universitario ed in tutte le mie esperienze.
Infine mi piacerebbe nominare alcune delle fantastiche persone che ho avuto la fortuna di conoscere e che mi hanno (e continuano a farlo) lasciato bellissime idee e influenze. Speciali ringraziamenti a:
Giacomo per avermi fatto prendere un bello spavento e avermi mostrato che con la determinazione tutto è possibile (e chissà cos'altro);
Gianluca per essere sempre al mio fianco quando si tratta di uscire dall'ordinario ed andare incontro allo stra-ordinario;
Filippo per essere costante prova del fatto che il mondo è grande e che allo stesso tempo paradossalmente le distanze possono sparire in un secondo;
Francesco e a tutti i **Boys** con i quali non è possibile stare seri e con cui è buona norma condividere sorrisi e avventure;
Bfs e Deja per avermi fatto presente che se qualcosa non c'è la si può creare;
Gaia per essere stata un vero e proprio vulcano di idee ed avermi insegnato il valore del sorriso;
Lorenzo e **Michele** per le sempre stimolanti discussioni dentro e fuori dall'università;
e a proposito di università non posso non ringraziare **Marcello**, grandissimo compagno di viaggio e di storie, **Francesco e Martina**, la spumeggiante coppia che scoppia, e poi ancora Eric, Elettra, Anna, Lucrezia, Federico, Fabri, Gabbo, Rocco, Mery e tutti gli altri: è stata una bella avventura.

Infine vorrei ringraziare mio fratello, con cui quotidianamente mi misuro, e mia madre per avermi insegnato ad essere quello che sono nel viaggio più lungo e strano che ci sia: la vita.

Bologna, 2025
Fabio Giordani