Introduzione

Il concetto di "gioco" è universale, presente in tutte le culture e periodi storici. Per astrarre questo concetto, si pensi ad esempio al gioco degli scacchi, del poker e del calcio: questi giochi sono sicuramente molto diversi tra loro, ma è possibile identificare alcune caratteristiche comuni. Ad esempio, essi sono costituiti da uno o più giocatori che possono eseguire varie azioni; ogni giocatore decide l'azione sulla base di informazioni cui ha accesso ed ottiene un guadagno per possibili combinazione di azioni dei vari giocatori. Lo studio dei giochi richiede quindi di modellare e analizzare le interazioni cooperative/conflittuali tra soggetti che prendono decisioni, avendo generalmente diversi orientamenti e obbiettivi.

Molti settori dell'informatica traggono beneficio dalla teoria dei giochi, ed in particolare l'intelligenza artificiale, la sicurezza, il calcolo distribuito. Non è quindi stupefacente la presenza di sinergie tra teoria dei giochi e complessità computazionale: da un lato nel disegno di procedure di decisione in giochi si evidenziano aspetti di complessità, dall'altro si rileva l'importanza del concetto di gioco nella comprensione della potenza e dei limiti dei vari meccanismi di calcolo. In questa nota ci si limiterà ad evidenziare alcuni di questi elementi prendendo in considerazione aspetti dei "giochi combinatori" e di semplici "giochi economici".

In un "gioco combinatorio" due giocatori modificano lo stato di un sistema applicando azioni (muovendo) alternativamente con regole predeterminate; il gioco parte da un fissato stato iniziale e termina quando uno dei giocatori finisce in condizione "di vincita". In ogni passo entrambe i giocatori hanno informazione completa sullo stato del sistema e l'obbiettivo di ognuno è di arrivare in uno stato di "vincita". La teoria dei giochi combinatori ha mosso i primi passi all'inizio del 1900 (L.C.Bouton, dell'Università di Harvard, trova la strategia vincente per il gioco del Nim nel 1901), ha raggiunto maturità negli anni trenta coi lavori di R.P.Spague (1935) e P.M.Grundy (1939) ed è stata sistematizzata da E.R. Berlekamp e J.H. Conway (libro di riferimento è E.R. Berlekamp, J.H. Conway, R.K. Guy, "Winning ways for your mathematical plays" , **I–II** , Acad. Press, 1982)

In un "gioco economico" i due giocatori giocano invece simultaneamente, scegliendo ognuno una sua "strategia"; ogni giocatore si ritrova poi un "guadagno" che dipende dalla coppia di strategie contemporaneamente scelte. L'obbiettivo di ogni giocatore è di garantirsi il massimo guadagno contro ogni strategia dell'avversario, senza tuttavia conoscerne la scelta. Questo approccio è proposto nel testo seminale di John von Neumann and Oskar Morgenstern, Theory of Games and Economic Behavior, 1944.

Nel paragrafo 1 si introducono i giochi combinatori e la loro classificazione in giochi imparziali (i due giocatori hanno lo stesso insieme di mosse) e partigiani. Si introduce il concetto di strategia vincente e si esibisce un algoritmo, lineare nella dimensione del grafo rappresentante il gioco, per decidere l'esistenza di una strategia vincente in gioghi imparziali senza ripetizione di mosse. Per tali giochi viene poi introdotta l'importante nozione di funzione di Sprague-Grundy (paragrafo 2) mostrando che anch'essa può essere calcolata in tempo lineare sequenziale; si dà invece evidenza che questa funzione non può essere calcolata con algoritmi paralleli veloci.

Nel paragrafo 3 è mostrato un algoritmo polinomiale per decidere 4 è dedicato all'estensione della funzione di Grundy al caso di giochi imparziali con ripetizione di mosse: per somme di tali giochi è possibile determinare in tempo polinomiale l'esistenza di una strategia vincente.

Il gioco degli scacchi avviene su una scacchiera 8x8, ma il grafo associato al gioco ha più di 10^{20} vertici. Descrivendo in maniera implicita le mosse legali degli scacchi, il gioco ha una rappresentazione estremamente più compressa del grafo che lo descrive esplicitamente: questa è la situazione comune a molti giochi, alcuni dei quali richiamati in paragrafo 5, ed è ragionevole che il tempo di calcolo di un eventuale algoritmo che determina l'esistenza di strategie vincenti venga riferito alla dimensione della rappresentazione compressa.

Ulteriormente, in paragrafo 5 sono considerate classi di giochi descritte da grafi con infiniti nodi (le configurazioni del gioco), in cui vale che, fissata una arbitraria configurazione iniziale, solo un numero finito di configurazioni è accessibile da essa. Descrivendo implicitamente il grafo infinito, il singolo gioco viene istanziato specificando solo la configurazione iniziale, la cui dimensione rappresenta la "dimensione compressa" del gioco: per lo studio di questi giochi è possibile utilizzare strumenti di complessità asintotica. A tal riguardo in paragrafo 6 sono richiamati i principali concetti sui modelli di calcolo deterministici, non deterministici e alternanti, unitamente alle principali classi di complessità e alle loro relazioni. Particolare attenzione è dedicata infine alla relazione tra il calcolo alternante e la determinazione di strategie vincenti in giochi, argomento ripreso in paragrafo 8 dedicato all'esposizione di giochi in PSPACE e EXPTIME.

Un importante concetto utile alla classificazione di problemi mediante strumenti di complessità computazionale è quello di completezza. A tale nozione è dedicato il paragrafo 7, in cui sono esibiti alcuni problemi NP-completi, PSPACE-completi, EXPTIME-completi, mentre giochi PSPACE-completi e EXPTIME-completi sono introdotti rispettivamente nei paragrafi 9 e 10.

1. Giochi combinatori e giochi imparziali senza ripetizione di mosse

Un *gioco combinatorio* è un gioco che soddisfa le seguenti condizioni:
1. Ci sono due giocatori (1e 2)
2. C'è un insieme (che considereremo finito) di possibili posizioni del gioco che chiameremo *stati*.
3. Le regole del gioco specificano, per ogni stato e ogni giocatore, quali possibili stati futuri possono essere raggiunti; la mossa di un giocatore consiste nel scegliere uno degli stati futuri legali. Se le regole non dipendono dal giocatore, il gioco è detto *imparziale*, altrimenti è detto *partigiano*.
4. I due giocatori alternano le loro mosse
5. Il gioco termina quando non ci sono più mosse possibili.

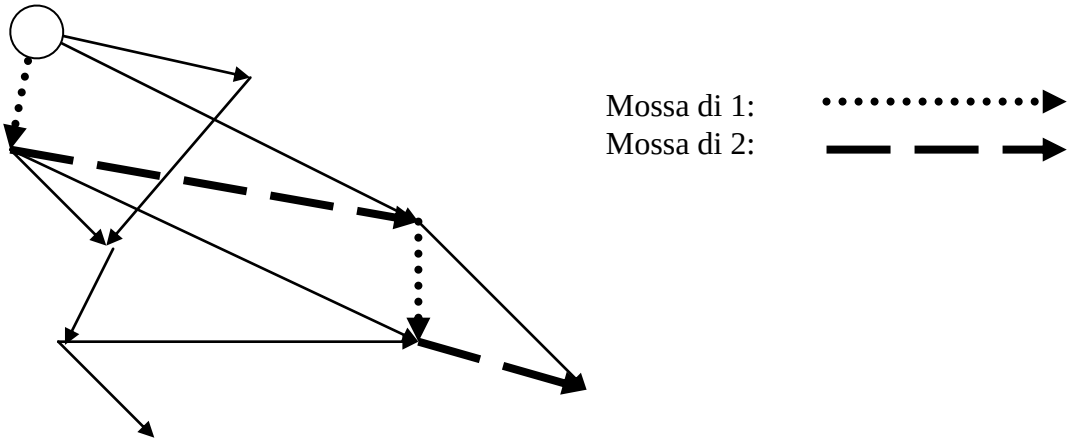
Si osservi che nella precedente definizione i due giocatori hanno completa conoscenza dello stato del gioco e non sono previste mosse simultanee o casuali.

In questa sezione ci si limita a analizzare il caso dei *giochi imparziali*: i due giocatori hanno completa informazione sullo *stato* del gioco e le possibili mosse da uno *stato* sono le stesse per i due giocatori.

Un gioco imparziale può essere formalmente descritto da tripla <S,A,s> dove S è l'insieme degli stati del gioco, $A \subseteq SxS$ è l'insieme di mosse legali da uno stato all'altro e $s \in S$ è lo stato iniziale; qui richiederemo ulteriormente che il grafo diretto G = <S,A> sia aciclico. Inizialmente viene posta una marca sullo stato iniziale e, alternativamente, il giocatore 1 e il giocatore 2 spostano la marca dal vertice corrente a uno dei vertici ad esso collegati mediante un arco. Il gioco termina quando la marca raggiunge un pozzo, cioè un vertice senza archi in uscita; il numero di mosse è detto *lunghezza* della partita. Con la regola di gioco detta "*normale*", vince il giocatore che ha mosso per ultimo, mentre con la regola di gioco detta "*misère*" perde il giocatore che muove per ultimo.

Esempio 1: Il gioco del poker non è imparziale poiché nessuno dei due giocatori ha piena informazione sullo stato del gioco (ognuno vede solo le sue carte). Nel gioco degli scacchi, invece, ogni giocatore ha piena informazione sullo stato, cioè le posizioni dei pezzi sulla scacchiera; tuttavia tale gioco non è imparziale perché, data una configurazione, le mosse possibili dei due giocatori sono diverse (ognuno può muovere solo pezzi del suo colore).

Esempio 2:



Una partita di lunghezza 4 del gioco, rappresentato dal grafo aciclico sopra disegnato in cui è

evidenziato lo stato iniziale, è il cammino segnalato in cui le mosse dei due giocatori hanno tratteggiatura diversa. In questo caso vince il giocatore 2 .

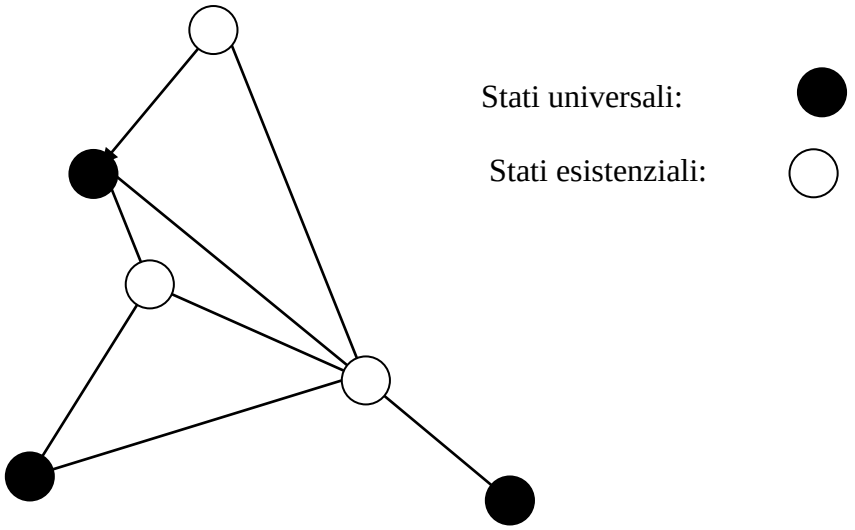
Esempio 3: *Giochi "sottrazione*". Si fissi un insieme di interi M e un intero n. Un gioco sottrazione G(M,n) è un gioco <S,A,s> in cui gli stati sono gli interi 0,1,..,n (cioè S = {0,1,..,n }), è possibile muovere dallo stato k allo stato j se k>j e k-j∈M (cioè A = {(k,k-d) | k∈{0,1,..,n }, k>d, d∈M), lo stato iniziale è n.

Esempio 4: *Gioco di Grundy*. Dato un intero n, il gioco di Grundy G(n) ha come stati i vettori di interi positivi di qualsiasi dimensione, col vincolo che la somma delle loro componenti è n; lo stato iniziale è il vettore di una sola componente (n), una mossa consiste nel scegliere una componente v_k del vettore e sostituirla con due componenti a_k e b_k la cui somma è v_k e tali che $a_k \neq b_k$. Il gioco termina quando tutte le componenti del vettore valgono 1 o 2.

Una nozione centrale nei giochi imparziali è quella di strategia vincente, che qui introduciamo nel caso di gioco "normale". Una *strategia vincente* per il giocatore 1 è ottenuta partizionando gli stati in due classi (stati universali/stati esistenziali), in modo tale tale che:

1) Lo stato iniziale è esistenziale
2) Gli stati pozzo sono universali
3) Se u è uno stato universale, ogni arco uscente porta a un stato esistenziale
4) Se e è uno stato esistenziale, almeno un arco uscente porta ad uno stato universale

Se un gioco <S,A,s> ammette una strategia vincente, allora il giocatore 1 può forzare sempre la vittoria al gioco. Basta che 1 applichi la regola che, se si trova in uno stato esistenziale, allora sceglie un arco che porta in uno stato universale. Poiché qualsiasi mossa faccia 2 da uno stato universale porta in uno stato esistenziale, 1 si troverà sempre in uno stato esistenziale e 2 in uno universale, quindi 1 vince poiché, essendo il grafo delle mosse finito e aciclico, prima o poi 2 sarà costretto a entrare in un pozzo.



Esempio di strategia vincente

Il seguente algoritmo, applicato a grafi aciclici G = <S,A> , permette di determinare gli stati esistenziali e universali del grafo G. Se lo stato iniziale s è uno stato esistenziale, questo algoritmo permette quindi al giocatore 1 di individuare una strategia vincente.

Ingresso: un grafo diretto aciclico G = <S,A>
Strutture dati:
U,I: vettori di insiemi di vertici, indiciati con vertici
C : coda di vertici
L : vettore booleano, indiciato con vertici

1. For all $y \in S$ **do** $U[y] = \{s' | (y,s') \in A\}$
2. For all $y \in S$ **do** $I[y] = \{s'' | (s'',y) \in A\}$
C = empty queue
3. For all $y \in S$, y pozzo **do** L(y) = 0 ; C = enqueue(y,C)
4. While $C \neq$ empty queue **do**
x = dequeue C
a. if $L[x]=0$ **then for** all $v \in I[x]$ **do**
b. L[v] = 1 ; C = enqueue(v,C)
c. if $L[x]=1$ **then for** all $v \in I[x]$ **do**
$U[v]=U[v]- \{x\}$;
d. If $U[v] = \Phi$ **then** L(v) = 0 ; C = enqueue(v,C)

Risposta: il vettore booleano L

L' algoritmo lavora in tempo O(|S|+|A|). Inoltre il vettore booleano in uscita L separa gli stati esistenziali (quelli per cui L[x] =1) da quelli universali (quelli per cui L[x] =0).

La correttezza dell'algoritmo può essere provata dalle seguenti osservazioni:

1- Un vertice viene introdotto nella coda solo se è stato precedentemente etichettato con 0 o con 1 (vedi linee **3., b., d.** dell'algoritmo)
2- Se un vertice v è etichettato con 1 (linea **b.**) allora esiste un suo successore x con L[x]=0 (vedi linea **a.**)
3- Se un vertice v è etichettato con 0, allora è un pozzo (linea **3.**) oppure l'insieme U[v] dei suoi successori è stato svuotato (linea **d.**). Ma un suo successore x viene cancellato da U[v] solo se L[x]=1 (vedi linea **c.**), quindi tutti i successori x di v sono tali che L[x]=1.

2. Funzione di Sprague-Grundy

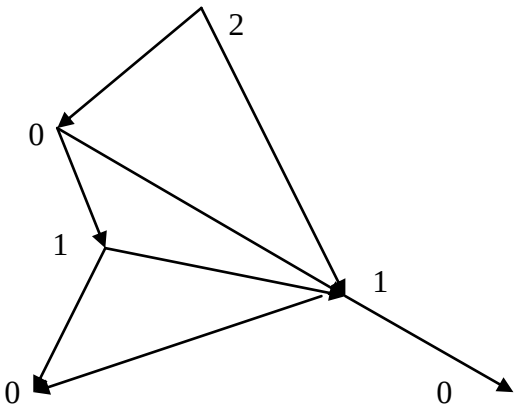
Una importante nozione, legata al concetto di strategia vincente in giochi imparziali, è quella di funzione di Grundy. . Essa viene anche chiamata chiamata funzione di Sprague-Grundy, essendo stata introdotta in modo indipendente da Grundy (P.M. Grundy, Mathematics and games , *Eureka* , **2,** pp. 6–8 , 1939)e Sprague (R. Sprague , Über mathematische Kampfspiele. *Tôhoku Math. J.* , **41** , pp. 438–444, 1935/36). In questa sezione introduciamo tale funzione, proviamo che può essere calcolata in tempo lineare nella dimensione del grafo associato al gioco ed evidenziamo che non esistono algoritmi paralleli efficienti per il suo calcolo (a meno di collassi di classi di complessità).

Sia N = {0, 1, 2, 3, …..} l'insieme degli interi non negativi. La *funzione di Grundy* di un grafo diretto aciclico G=<S,A> è una funzione f_G che ad ogni vertice in S associa un intero non negativo; $f_G(v)$ è detto valore di Grundy del vertice v. Il valore di Grundy di v è 0 se v è un pozzo, altrimenti è il più piccolo intero che non appare tra i valori di Grundy di vertici raggiunti da archi uscenti da v.

Formalmente, se $S \subseteq N$, definiamo Mes S il più piccolo intero non presente in S, cioè

$$\text{Mes } S = \text{Min } N\text{-}S$$

Mediante l'operazione Mes, la funzione di Grundy può essere ricorsivamente definita ponendo $f_G(v)= 0$ se v è un pozzo, $f_G(v)= \text{Mes } \{ f_G(s) | (v,s) \in A\}$ altrimenti.



Esempio di funzione di Grundy: si osservi che essa assume valore 0 negli stati universali

Essa può essere calcolata dal seguente algoritmo:

Ingresso: un grafo diretto aciclico <S,A>

(1) Per ogni pozzo p poni L(p)=0 e dichiara p etichettato
(2) **Se** u è un vertice non etichettato per cui $\{v \mid (u,v) \in A\}$ contiene solo vertici etichettati, **allora** poni $L(v) = \text{Mes } \{L(v) | (u,v) \in A\}$ e dichiara u etichettato. **Ripeti** (2)
(3) Per ogni v etichettato poni $f_G(v) = L(v)$

Una implementazione più dettagliata è la seguente:

Ingresso: un grafo diretto aciclico G = <S,A>
Strutture dati:
U,I: vettori di insiemi di vertici, indiciati con vertici
E : vettore di insieme di interi, indiciato con vertici
C : coda di vertici
L : vettore di interi, indiciato con vertici
For all y∈S **do** U[y] = {s'| (y,s') ∈A}
For all y∈S **do** I[y] = {s''| (s'',y) ∈A}
For all y∈S **do** E [y] = Φ
C = empty queue
For all y∈S, y pozzo **do** L(y) = 0 ; C = enqueue(y,C)
While C≠ empty queue **do**
x = dequeue C
for all v∈I[x] **do**
U[v]=U[v]- {x};
If U[v] = Φ **then** L(v) = Mes E[v] ; C = enqueue(v,C)
Risposta: il vettore di interi L

Una semplice analisi permette di provare che l'algoritmo precedente è corretto e lavora in tempo O(|S|+|A|). Vale quindi:

Proposizione 1: Esiste un algoritmo sequenziale che calcola la funzione di Grundy di un grafo diretto e aciclico G=<S,A> in tempo O(|S|+|A|).

Se l'ingresso all'algoritmo sopra delineato è esplicitamente il grafo <S,A>, la funzione di Grundy è calcolabile in tempo polinomiale (addirittura in tempo lineare).

Per molti giochi tuttavia l'istanza <S,A> è data in forma compressa, con una dimensione pari a log(|S|+|A|): una classe di tali problemi sarà esposta la prossima sezione attraverso il concetto di "somma di giochi", di cui il gioco del Nim rappresenta un importante esempio. L'applicazione diretta dell'algoritmo precedente porta generalmente, in questo caso, tempi di calcolo esponenziali per il semplice fatto che il numero di vertici del grafo può essere esponenziale rispetto alla sua rappresentazione compressa. E' di grande importanza in questo caso risolvere il problema che, avendo come ingresso la rappresentazione compressa di un gioco e un suo vertice, richiede come risposta il solo valore della funzione di Grundy su quel vertice.

Esempio 1: consideriamo il "gioco sottrazione" G({1,2,3}, n). Esso ha come stati l'insieme {1,2,…,n}; lo stato 1 è collegato allo stato 0, che è l'unico pozzo, lo stato 2 è collegato agli stati 0 e 1, per j>2 lo stato j è collegato agli stati j-1, j-2, j-3. Lo stato iniziale del gioco è n. Per ispezione diretta si verifica che $f_G(0)=0$, $f_G(1)=1$, $f_G(2)=2$, $f_G(3)=3$, $f_G(4)=0$, $f_G(5)=1$, $f_G(6)=2$, $f_G(7)=3$, $f_G(8)=0$. Questo prova che la funzione è periodica di periodo 4, così che in generale $f_G(j)= <j>_4$. Si osservi che, rappresentando in binario il numero n, il gioco G({1,2,3}, n) è essenzialmente rappresentabile con log n bit, quindi il numero di vertici del grafo, che è n+1, è esponenziale rispetto alla dimensione log n. Ciò nonostante, il valore $<j>_4$ della funzioni di Grundy sul vertice j è calcolabile in tempo lineare in log n. □

Esempio 2: la funzione di Grundy del "gioco sottrazione" G({1,2,…,k}, n) è $f_G(j) = \langle j \rangle_{k+1}$; la funzione risulta quindi periodica. Più in generale, è facile osservare che la funzione di Grundy di un gioco sottrazione G(M,n), dove M è un insieme finito, risulta periodica. □

Esempio 3: un circuito di Grundy è una sequenza di istruzioni $I_1, I_2, \ldots, I_b$, dove l'istruzione I_k è del tipo $Z_k = Mes(Z_i, Z_s) / Mes(Z_i) / 0$, con $i,s<k$. In un circuito di Grundy ogni variabile Z_k assume un valore intero $w(Z_k)$ definito induttivamente:

- Se l'istruzione I_k è $Z_k = 0$ allora $w(Z_k)=0$
- Se l'istruzione I_k è $Z_k = Mes(Z_i)$ allora $w(Z_k)= Mes(w(Z_i))$
- Se l'istruzione I_k è $Z_k = Mes(Z_i, Z_s)$ allora $w(Z_k)= Mes(w(Z_i), w(Z_s))$

Dato un circuito di Grundy, possiamo costruire un gioco dato dal grafo aciclico G=<S,A>, dove $S=\{ Z_1,\ldots, Z_b\}$ è l'insieme delle variabili. Z_k è un pozzo se l'istruzione I_k è $Z_k = 0$, mentre Z_k è collegata a Z_i se l'istruzione I_k è $Z_k = Mes(Z_i)$ e Z_k è collegata a Z_i e a Z_s se l'istruzione I_k è $Z_k = Mes(Z_i, Z_s)$. E' facile verificare per induzione che la funzione di Grundy di G è $f_G(Z_k)= w(Z_k)$ □

Abbiamo visto che la funzione di Grundy di un grafo è calcolabile con un algoritmo sequenziale efficiente, che lavora essenzialmente in tempo lineare. Mostriamo ora che non ci si può invece aspettare che possa essere calcolata efficientemente da una macchina parallela. Richiamiamo rapidamente lo strumento generalmente usato per trattare questa problematica.

Con NC si denota la classe dei problemi risolubili su pRAM in tempo polilogaritmico ($\log^{O(1)}n$) e con un numero polinomiale di processori ($n^{O(1)}$) e con P si denota la classe di problemi risolubili in tempo polinomiale con algoritmi sequenziali. E' noto che NC⊆P; P è comunemente assunto come l'insieme dei problemi "efficientemente risolubili su macchina sequenziale" (tesi di Church estesa), mentre NC identifica la classe dei problemi "efficientemente risolubili su macchina parallela". Un problema X è detto P-completo se X è in P e se ogni problema in P è riducibile a X, via una riduzione realizzabile su pRAM in tempo polilogaritmico e con un numero polinomiale di processori. Ne segue che se un problema P-completo X fosse in NC, allora sarebbe P=NC. Ipotizzando la congettura (generalmente accettata dagli esperti, ma attenzione agli esperti …) che NC≠P, possiamo quindi concludere che un problema P-completoX non può essere in NC, quindi X non ammette un algoritmo parallelo efficiente.

Un classico problema P-completo è CIRCUIT-VALUE:

Problema: CIRCUIT VALUE
Istanza: Un circuito booleano Φ (con variabili $x_1,.., x_n$ e istruzioni del tipo $Z_k = Z_i \wedge Z_j$ / not Z_i/ x_s con $i,j<k$) ed un assegnamento $(a_1,.., a_n)$ alle variabili $x_1,.., x_n$
Questione: E' f(a)=1, dove f è la funzione booleana calcolata da Φ?

Il problema CIRCUIT VALUE può essere ridotto al calcolo della funzione di Grundy. L'osservazione di base è che, se x e y sono variabili booleane, allora si può verificare che

(*) $\quad x \wedge y = Mes(Mes(x,y),Mes(x,y))$, $\quad$ not $y = Mes(y)$

Assegnato ora un circuito Φ ed un assegnamento $(a_1,.., a_n)$, possiamo costruire un circuito di Grundy Ψ (vedi **Esempio 3**) come segue:

- l'istruzione $Z_k = Z_i \wedge Z_j$ viene sostituita da $Z_k = Mes(Z'_k, Z'_k)$; $Z'_k = Z_i \wedge Z_j$
- l'istruzione $Z_k =$ not Z_i viene sostituita da $Z_k = Mes(Z_i)$
- l'istruzione $Z_k = x_s$ viene sostituita da $Z_k=0$, se $a_s=0$, oppure da $Z_k=Mes(Z'_k)$;$Z'_k=0$, se $a_s=1$

E' facile costruire un efficiente algoritmo parallelo che, avendo in ingresso l'istanza $\langle \Phi,(a_1,.., a_n)\rangle$ di CIRCUIT VALUE, dà in uscita il circuito di Grundy Ψ; inoltre, per costruzione e a causa

dell'identità (*), il valore calcolato da Ψ coincide col valore di Φ sull'assegnamento (a_1,.., a_n). Se quindi esistesse un efficiente algoritmo parallelo per il calcolo della funzione di Grundy, saremmo in grado di risolvere efficientemente in parallelo il problema CIRCUIT VALUE, che è un problema P-completo!

La riduzione precedente, unitamente al fatto che la funzione di Grundy di un grafo diretto aciclico si può calcolare in tempo polinomiale, prova che la determinazione della funzione di Grundy è un problema P-completo e quindi:

Proposizione 2: Se NC≠P allora la funzione di Grundy di grafi diretti aciclici non può essere calcolata da alcun algoritmo parallelo efficiente (realizzabile su pRAM in tempo polilogaritmico e con un numero polinomiale di processori).

Ritornando al problema generale dei giochi imparziali, la conoscenza della funzione di Grundy permette di determinare in modo costruttivo l'esistenza di una strategia vincente del giocatore 1, come dato da:

Proposizione 3: se f_G è la funzione di Grundy del grafo diretto G=<S,A>, allora $\{v|f_G(v)=0\}$ è l'insieme degli stati universali, mentre $\{v| f_G(v)\neq 0\}$ è l'insieme degli stati esistenziali. Pertanto il giocatore 1 ha una strategia vincente nel gioco <S,A,s> se e solo se $f_G(v)\neq 0$.
Dimostrazione: osserviamo che $f_G(x) = 0$ se e solo se o x è un pozzo oppure per ogni v con $(x,v)\in A$ vale che $f_G(v)\neq 0$; analogamente vale che $f_G(x) \neq 0$ se e solo se per ogni v con $(x,v)\in A$ vale che $f_G(v) = 0$ □

Esempio 4: sia g(n) il valore della funzione di Grundy del "gioco di Grundy" G(n) (vedi sez.1) sullo stato iniziale n. La seguente tabella presenta i primi 15 valori di g(n):

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
g(n)	0	0	1	0	2	1	0	2	1	0	2	1	3	2	1

Ad esempio, in base a **Proposizione 3** nel gioco G(15) il giocatore 1 può forzare l'avversario a finire in un pozzo, ciò che non è vero per il gioco G(10).
Sono noti i valori di g(n) fino ad $n=2^{35}$, ma è ancora un problema aperto se la funzione g(n) sia definitivamente periodica. □

Come abbiamo visto sopra, la funzione di Grundy contiene sicuramente l'informazione sugli stati esistenziali e universali del gioco, e qualcosa di più. Nella prossima sezione vediamo l'utilità effettiva di questa informazione aggiunta, che permetterà il calcolo del valore della funzione di Grundy su un dato vertice (e quindi l'individuazione di una strategia vincente effettiva) anche per molti giochi espressi in forma compressa attraverso l'operazione di somma di giochi.

3. Somma di giochi

Una importante operazione tra giochi, capace di generare nuovi giochi partendo da giochi preassegnati, è quella di somma. Questa operazione permette di rappresentare in forma compressa una sottoclasse di giochi in cui l'esistenza di una strategia vincente è risolubile in tempo polinomiale.

Informalmente, dati due giochi imparziali $G_1 =< S_1, A_1, s_1>$ e $G_1 =< S_2, A_2, s_2>$, la somma $G_1+ G_2$ è il gioco in cui il giocatore di turno sceglie liberamente un gioco tra G_1 e G_2 e muove con le regole del gioco scelto, lasciando inalterato lo stato dell'altro. Il gioco termina quando non è possibile nessuna mossa in G_1 o in G_2 .

Più formalmente, $< S_1, A_1, s_1> + < S_2, A_2, s_2>$ è il gioco descritto da $< S_3, A_3, s_3>$ dove $S_3=S_1xS_2$, s_3 è la coppia di stati s_1s_2, $A_3 = \{(sq,pq)|\ (s,p)\in A_1, q\in A_2\}\cup\{(rm,rz)|\ (m,z)\in A_2, r\in A_1\}$.

Identificando la dimensione di un gioco <S,A,s>, dato esplicitamente, con il numero di stati |S|, osserviamo che la dimensione della somma di giochi di dimensione n_1 e n_2 è il prodotto $n_1.n_2$; ne segue che la somma di n giochi di dimensione c è un gioco di dimensione c^n. Osserviamo tuttavia che la somma di n giochi può essere rappresentata "implicitamente" elencando gli n giochi, con una occupazione di O(c.n) bit. Questa rappresentazione è pesantemente compressa: in generale l'uso dell'algoritmo che abbiamo delineato per il calcolo della funzione di Grundy richiede di decomprimere la rappresentazione dell'ingresso, e porta a tempi di calcolo esponenziali.

Fortunatamente la funzione di Grundy di una somma di giochi si calcola facilmente una volta nota la funzione di Grundy dei singoli giochi, e questo porta ad un algoritmo polinomiale per la determinazione di una strategia vincente per una somma di giochi, espressa in forma compressa

A questo riguardo, introduciamo l'operazione $\oplus$: $NxN\rightarrow N$ detta "somma Nim", dove $a\oplus b$ è la operazione di "or esclusivo" bit per bit applicata alla rappresentazione binaria dei numeri a,b. La definizione formale è la seguente:

Definizione 1: dati due interi a,b le cui rappresentazioni binarie sono $a_1 ..a_m$ e $b_1 ..b_m$ allora la loro *somma Nim* è $a\oplus b=c$, dove la rappresentazione binaria di c è $c_1..c_m$ con $c_k= a_k\oplus b_k$ $(1\leq k\leq m)$.

Esempio 1: $3\oplus4=7$ (infatti $011\oplus100=111$) $6\oplus5= 3$ (infatti $110\oplus101=011$)

La somma Nim è associativa e commutativa; inoltre vale che $a\oplus a=0$ per ogni a. Una immediata conseguenza è che, se $a\oplus b=c$, allora $a= c\oplus b$. Una ulteriore proprietà è la seguente:

Lemma 1: supponiamo che $b= u\oplus s$ e $b>a$; posto $d= a\oplus b$, allora o $d\oplus u<u$ oppure $d\oplus s<s$.
Dimostrazione : identifichiamo con abuso di linguaggio i numeri a,b,u,s,d con le loro rappresentazioni binarie con un fissato numero m di bit.
Poiché b>a, sarà, per un opportuno k, $b=b_1 .. b_{k-1}1b_{k+1} ..b_m$ mentre $a=b_1 .. b_{k-1}0a_{k+1} ..a_m$; di conseguenza $d = a\oplus b = 00..01d_{k+1}..d_m$. Poiché $u_k\oplus s_k = b_k =1$, sarà o $u_k =1$ oppure $s_k =1$. Supponiamo senza perdita di generalità che sia $u_k =1$. Segue che i primi k bit di u sono $u_1 .. u_{k-1}1$, mentre i primi k bit di $d\oplus u$ sono $u_1 .. u_{k-1}0$, e quindi $d\oplus u<u$.

Il seguente elegante risultato, centrale nella teoria sviluppata da Grundy e Sprague, permette di esprimere la funzione di Grundy di una somma di giochi in termini delle funzioni di Grundy dei giochi presi separatamente.

Proposizione 1: $f_{G1+G2}(pq) = f_{G1}(p) \oplus f_{G2}(q)$, dove pq è una coppia di vertici p e q dei grafi G_1 e G_2

Dimostrazione: Sia il grafo somma $G=G_1+ G_2$ descritto dal grafo diretto aciclico <S,A> e sia $g(qp) = f_{G1}(q) \oplus f_{G2}(p)$. Per mostrare che g è la funzione di Grundy f_{G1+G2} del grafo G basta provare le seguenti tre proprietà:

1. se v è un pozzo di G, allora $g(v)=0$
2. se per $v \in S$ è $g(v)=b>a\geq 0$, allora esiste in A un arco (v,w) con $f_G(w)=a$
3. se per $v \in S$ è $g(v)=b$ e (v,w) è un arco in A, allora $g(v) \neq g(w)$

Proviamo le tre proprietà separatamente.

1. Se v=qp è un pozzo allora p e q sono pozzi in $G_1+ G_2$, quindi $g(v) = f_{G1}(q) \oplus f_{G2}(p)=0 \oplus 0=0$

2. Sia $g(v)=b>a\geq 0$ con v=qp e quindi $b= f_{G1}(q) \oplus f_{G2}(p)$. Poniamo $d=a \oplus b$ da cui segue $a=d \oplus b$. Senza perdita di generalità, per il **lemma 1** possiamo supporre che $d \oplus f_{G1}(q) < f_{G1}(q)$. Poiché f_{G1} è funzione di Grundy di G_1, $f_{G1}(q)= Mes \{ f_{G1}(s) | (q,s) \in A_1\}$ ed essendo $d \oplus f_{G1}(q) < f_{G1}(q)$ esiste un arco $(q,q') \in A_1$ tale che $d \oplus f_{G1}(q) = f_{G1}(q')$. Abbiamo allora trovato un vertice w=q'p nel grafo G per cui $g(w)= f_{G1}(q') \oplus f_{G2}(p)= d \oplus f_{G1}(q) \oplus f_{G2}(p)= d \oplus b=a$.

3. Sia v=qp e, senza perdita di generalità, w=q'p con (q,q') arco in A_1. Se fosse g(v)=g(w), sarebbe $f_{G1}(q) \oplus f_{G2}(p)= f_{G1}(q') \oplus f_{G2}(p)$ e quindi $f_{G1}(q)=f_{G1}(q')$ con (q,q') arco in A_1, assurdo poiché f_{G1} è funzione di Grundy del grafo G_1. □

Dato un gioco $G= G_1+..+ G_m$ che è somma di m giochi $G_1 = <S_1, A_1, s_1>, \ldots , G_m = <S_m, A_m, s_m>$, il precedente risultato ci permette di decidere l'esistenza di una strategia vincente per G col seguente algoritmo:

Ingresso: m giochi $G_1 , G_2 ,\ldots, G_m$
For k=1,m **do** calcola $g_k= f_{Gk}(s_k)$
If $g_1 \oplus \ldots \oplus g_m \neq 0$ **then** il giocatore 1 ha una strategia vincente
else il giocatore 1 ha non una strategia vincente

Il tempo di calcolo è $O(|A_1|+ |A_2|+ \ldots+|A_m|)$, che è essenzialmente lineare nella dimensione della rappresentazione compressa del gioco somma.

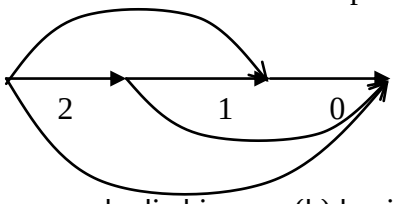
Il giocatore 1, se ha individuato in tempo polinomiale l'esistenza di una strategia vincente, la può parimenti implementare in tempo polinomiale.

Supponiamo infatti che 1 debba muovere partendo da una configurazione di stati $v_{1..} v_m$ tale che $f_{G1}(s_1) \oplus \ldots \oplus f_{Gm}(s_m) \neq 0$. Allora per prima cosa calcola $d= f_{G1}(v_1) \oplus \ldots \oplus f_{Gm}(v_m)$; determina poi k ($1 \leq k \leq m$) e v'_k tale che (v_k,v'_k) è un arco in A_k ed inoltre $f_{Gk}(v'_k)= f_{Gk}(v_k)+d$. Muove quindi in $v_{1..} v'_{k..} v_m$ così che il giocatore 2 sia in una configurazione con $f_{G1}(v_1) \oplus \ldots f_{Gk}(v'_k) \oplus \ldots \oplus f_{Gm}(v_m) =0$

□

Esempio 1: *il gioco del Nim*

Questo è probabilmente il più noto gioco imparziale. Il gioco di base è la *pila Nim*, che è il "gioco sottrazione" G(N,n). Si parte con una pila di n gettoni, a turno i giocatori rimuovono quanti gettoni vogliono dall'intera pila e vince il giocatore che svuota la pila. Il gioco è descritto dal grafo con stati S= {n,n-1,…,2,1,0} e (i,j) è un arco se i>j. Si verifica facilmente che la funzione di Grundy della pila Nim è f(j)=j; ovviamente se la pila iniziale non è vuota il giocatore 1 vince banalmente svuotando immediatamente la pila.



(A fianco)
Grafo di una pila Nim di 3 elementi

Convenendo di chiamare (k) la pila Nim con k marche; il *gioco del Nim N* è una somma di m pile Nim:

$$N = (k_1)+ (k_2)+\ldots+ (k_m)$$

Se $k_1 \oplus k_2 \oplus \ldots \oplus k_m \neq 0$ allora il giocatore 1 può forzare il giocatore 2 alla sconfitta., in quanto può realizzare una strategia per cui muove su stati con funzione di Grundy diversa da 0, e costringe il giocatore 2 in stati con funzione di Grundy uguale a 0. Supponiamo infatti che in una certa fase del gioco sia il turno del giocatore 1 e che lo stato del gioco sia descritto da ($z_1, z_2,\ldots, z_m$) con $z_1 \oplus z_2 \oplus \ldots \oplus z_m \neq 0$. Posto $z = z_1 \oplus z_2 \oplus \ldots \oplus z_m$, il giocatore 1 determina un indice k tale che $z \oplus z_k < z_k$ (un tale indice esiste sempre come si vede da **lemma1**) e toglie $z_k - z \oplus z_k$ marche nella posizione k; poiché in posizione k sono restate $z \oplus z_k$ marche, il giocatore 2 è obbligato a muovere dallo stato $(z_1,z_2,.., z \oplus z_k \ldots, z_m)$ in cui vale $z_1 \oplus z_2 \oplus \ldots \oplus z \oplus z_k \oplus \ldots \oplus z_m = z_1 \oplus z_2 \oplus \ldots \oplus z_m \oplus z = z \oplus z = 0$.

Per lo stesso motivo, se invece nello stato iniziale vale che $k_1 \oplus k_2 \oplus \ldots \oplus k_m = 0$, il giocatore 2 può forzare il giocatore 1 alla sconfitta. □

Esempio 2: giochi che si decompongono come somme di "se stessi"

Una naturale applicazione della funzione di Grundy è la ricerca di strategie vincenti in giochi imparziali che in ogni fase possano essere interpretati "ricorsivamente" come somma dello stesso gioco su stati iniziali differenti.

Il gioco di Grundy ne è un prototipo. Supponiamo di partire dallo stato iniziale (n) del gioco G(n) e chiamiamo g(n) il valore della funzione di Grundy su tale stato; la prima mossa consiste nel decomporre n in due interi k,s con $0<k<n$, $k+s=n$, $k \neq s$, ottenendo lo stato (k,s). Il gioco che parte dallo stato (k,s) è allora G(k)+G(s): il valore della funzione di Grundy su tale stato è allora $g(k) \oplus g(s)$. Questo ci permette di ottenere la seguente equazione di ricorrenza per il calcolo di g(n):

g(1)=0
g(2)=0
$g(n)= Mes \{g(k) \oplus g(n-k) \mid 0<k<n, k \neq n-k\}$ se n>2

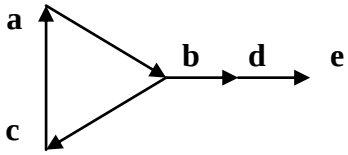
Il valore della funzione di Grundy su qualsiasi stato (k_1, .., k_j) è infine $g(k_1) \oplus \ldots \oplus g(k_j)$: se $g(n) \neq 0$ siamo in grado allora di esibire una strategia del gioco G(n) con cui il giocatore 1 forza l'avversario in un pozzo. □

4. Giochi imparziali con ripetizione di mosse

Un gioco imparziale è descritto da una tripla <S,A,s>, dove S è l'insieme degli stati del gioco, $A \subseteq SxS$ è l'insieme di mosse legali da uno stato all'altro e $s \in S$ è lo stato iniziale. Fino ad ora avevamo ulteriormente richiesto che il grafo diretto G = <S,A> fosse aciclico; in questo caso è stato possibile, per esempio, provare l'esistenza e unicità della funzione di Grundy, cosa che ha come conseguenza l'esistenza di una strategia vincente per uno dei giocatori.

Giochi con grafi aciclici non possono prevedere ripetizioni di mosse: in questa sezione consideriamo il caso più generale di grafi diretti che possono contenere cicli. Una importante conseguenza è l'esistenza di partite di durata infinita: mentre nel caso di grafi aciclici ogni partita è finita, quindi termina con un vincitore, qui è possibile che uno dei due giocatori, pur non potendo vincere, costringa l'avversario al pareggio. Gli esiti di una partita sono dunque tre: vince il giocatore 1, vince il giocatore 2, viene dichiarato pareggio.

Esempio 1: nel gioco rappresentato dal grafo a destra, in cui lo stato iniziale è **a**, il giocatore che da **b** muove in **d** perde. Entrambe i giocatori hanno allora interesse a realizzare la partita **a,b,c,a,b,c,….**



Abbiamo visto che il problema dell'esistenza di strategie vincenti in giochi descritti da grafi aciclici (o più in generale da somme di tali giochi), è ridotto agevolmente al calcolo della funzione di Grundy. Ora, per grafi diretti generali il concetto di funzione di Grundy perde il senso che era stato attribuito nel caso aciclico. Tale funzione può non essere unica, o addirittura può non esistere; il problema di decidere se un grafo diretto ammette una funzione di Grundy è NP-completo.

Esempio 2:

In Fig.1 è rappresentato un grafo che non ammette funzione di Grundy, mentre per il grafo in Fig.2 è possibile esibire due funzioni di Grundy.

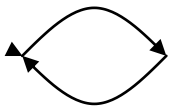


Fig.1

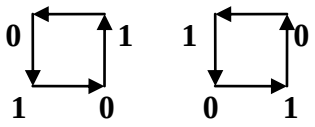


Fig.2

Fortunatamente il concetto di funzione di Grundy può essere esteso esteso al caso generale (vedi "Smith, C.A.B., Graph and composite games, J. Comb. Theory 1, 51-88, 1966), e a questa estensione è dedicato il resto di questa sezione.

Dato un grafo diretto G = <S,A> e un suo vertice u, con u^+ denotiamo l'insieme $\{v| (u,v) \in A\}$ dei "seguenti" di u. Considereremo funzioni $g:S \to N \cup \{\infty\}$, con $N= \{0,1,2, \ldots\}$e supponendo che $\infty > x$ per ogni $x \in N$; estendiamo la funzione g a sottoinsiemi $J \subseteq S$ ponendo $g(J)= \{g(v)| \ v \in J$ e $g(v) < \infty\}$. Dati due vertici v,z , diremo che
$g(v) \equiv g(z)$ se $g(v)< \infty$, $g(z)< \infty$ e $g(v)=g(z)$, oppure $g(v)=g(z)= \infty$ e $g(v^+)= g(z^+)$

Se $g(v) = g(z) = \infty$, perché sia $g(v) \equiv g(z)$ deve ulteriormente valere che $g(v^+)= g(z^+)$. E' allora conveniente usare la scrittura $g(v)= \infty(X)$, con $X= g(u^+)$, che evidenzia entrambe gli elementi che portano alla relazione $\equiv$. Dato G = <S,A> e una funzione $g:S \to N \cup \{\infty\}$, porremo infine $S^f=\{v|$ $g(v)< \infty\}$.

Definizione 1: dato un grafo diretto G = <S,A>, la *funzione di Sprague-Grundy generalizzata* (detta anche *γ-funzione*) con *funzione contatore* c: $S^f \to N$ è una funzione $\gamma: S \to N \cup \{\infty\}$ tale che:

a. Se $\gamma(u) < \infty$ allora $\gamma(u) = \text{Mes}\ \gamma(u^+)$
b. Se esiste $v \in u^+$ con $\gamma(v) > \gamma(u)$ allora esiste $z \in v^+$ con $\gamma(z) = \gamma(u)$ e $c(u) > c(z)$
c. Se per ogni $v \in u^+$ con $\gamma(v) = \infty$ esiste $z \in v^+$ con $\gamma(z) = \text{Mes}\ \gamma(u^+)$, allora $\gamma(u) < \infty$

Osserva che se l'ipotesi della condizione c. è soddisfatta, allora $\gamma(z) < \infty$; poiché per la condizione a. vale che $\gamma(u) = \text{Mes}\ \gamma(u^+)$, e quindi $\gamma(z) = \text{Mes}\ \gamma(u^+) = \gamma(u)) < \infty$.

Analizziamo ora il ruolo che assume la *funzione di Grundy generalizzata* nello studio delle strategie vincenti del gioco descritto dal grafo G=<S,A>. A tal riguardo, classifichiamo i nodi di S in tre categorie:

1. Un U-nodo è un nodo u in cui l'avversario del giocatore che muove da u ha una strategia vincente
2. Un E-nodo è un nodo u tale che il giocatore che muove da u ha una strategia vincente
3. Un D-nodo è un nodo u in cui nessuno dei giocatori che parta da u ha una strategia vincente, ma ogni giocatore può sempre trovare una mossa non perdente.

La precedente classificazione crea una partizione di S nelle tre classi *U,E,D* formate rispettivamente dai nodi di tipo U, E, D. Vale la seguente:

Proposizione 1: se γ è una funzione di Grundy del grafo <S,A>, allora:

- $U = \{u \mid \gamma(u) = 0\}$
- $D = \{u \mid \gamma(u) = \infty \text{ e } 0 \notin \gamma(u^+)\}$
- $E = \{u \mid 0 < \gamma(u) < \infty\} \cup \{u \mid \gamma(u) = \infty \text{ e } 0 \in \gamma(u^+)\}$
- Per ogni $u \in U$ e ogni $v \in u^+$ esiste $z \in v^+ \cap U$ con $c(u) > c(z)$

Dimostrazione: basta provare che:

a) se $\gamma(u) = \infty$ e $0 \notin \gamma(u^+)$, allora esiste $v \in u^+$ tale che $\gamma(v) = \infty$ e $0 \notin \gamma(v^+)$.
b) se $\gamma(u) = 0$ allora per ogni nodo $v \in u^+$ vale che o $0 < \gamma(v) < \infty$ oppure $\gamma(v) = \infty$ e $0 \in \gamma(v^+)$.
c) se $u \in \{y \mid 0 < \gamma(y) < \infty\} \cup \{y \mid \gamma(y) = \infty \text{ e } 0 \in \gamma(y^+)\}$ allora esiste $v \in u^+$ con $\gamma(v) = 0$.
d) se $\gamma(u) = 0$ e $v \in u^+$ allora esiste $z \in v^+$ con $c(u) > c(z)$

Proviamo la a). Supponiamo che $\gamma(u) = \infty$ e $0 \notin \gamma(u^+)$. Poiché $0 \notin \gamma(u^+)$ vale che $\text{Mes}\ \gamma(u^+) = 0$. Inoltre per la contronominale della condizione c. esiste $v \in u^+$ tale che $\gamma(v) = \infty$ e, per ogni $z \in v^+$, vale che $\gamma(z) \neq \text{Mes}\ \gamma(u^+) = 0$. Questo significa che $\gamma(v) = \infty$ e $0 \notin \gamma(v^+)$.

Proviamo la b) e la d). Sia infatti $\gamma(u) = 0$. Allora per la a. vale che $0 = \text{Mes}\ \gamma(u^+)$. Consideriamo un qualsiasi $v \in u^+$. Se $\gamma(v) < \infty$ allora $\gamma(v) > 0$, se invece $\gamma(v) = \infty$ allora per la b. esiste $z \in v^+$ con $\gamma(z) = \gamma(u) = 0$, cioè $0 \in \gamma(v^+)$. Poiché inoltre è $\gamma(v) > 0 = \gamma(u)$, sempre per la b. è $c(u) > c(z)$.

Proviamo la c). Sia $0 < \gamma(u) < \infty$; per a. vale che $0 < \gamma(u) = \text{Mes}\ \gamma(u^+)$, quindi esiste $v \in u^+$ con $\gamma(v) = 0$. Sia $\gamma(u) = \infty$ e $0 \in \gamma(u^+)$; esiste allora $v \in u^+$ con $\gamma(v) = 0$. □

Si osservi che il giocatore che muove da un vertice in *E* ha una mossa che porta in *U*. Se un giocatore muove invece da un vertice in *U* , qualsiasi mossa faccia porta in *E*. Si osservi inoltre che il giocatore partente da una configurazione in *E* può forzare l'altro in una sequenza di configurazioni $u_1, .., u_k, \ldots,$ con $c(u_{k+1}) < c(u_k)$ e tale sequenza deve necessariamente essere finita: il giocatore che parte da un vertice in *E* ha una strategia vincente. Si osservi infine che per ogni nodo in *D* c'è un nodo uscente ancora in *D*, ma da *D* non è possibile muovere in *E*. Il giocatore che parte

da una configurazione in *D* può sempre trovare una mossa non perdente, ma nemmeno ha una strategia vincente.

Un algoritmo per il calcolo della funzione di Grundy generalizzata è il seguente:

Ingresso: un grafo diretto <S,A>
Risposta: due vettori l(u), c(u) con $u \in S$

m = 0; i = 0; H = S

while $S \neq \varnothing$ **do**

1. **while** "esiste $u \in H$: $i \notin l(u^+) \wedge ((v \in u^+ \wedge (v \in H \vee l(v) = \infty)) \Rightarrow$ esiste $w \in u^+$ con $l(w)=i)$"
   do l(u) = i; c(u) = m; m = m+1; H = H/ {u}
2. **for all** $u \in H$ **do**
   if $i \notin l(u^+)$ **then** $l(u) = \infty$; H = H/ {u}
3. i = i+1

Il vettore l(u) determinato col precedente algoritmo associa ad ogni vertice u il valore $\gamma(u)$ della funzione di Grundy su u, mentre il vettore c(u) definisce il corrispondente contatore. Per grafi diretti <S,A>, il precedente algoritmo termina dopo O(|S|.(|S|.|A|)) passi di calcolo, quindi lavora in tempo polinomiale nella dimensione del grafo (anche se non più lineare, come nel caso di grafi aciclici).

Anche per giochi rappresentati da grafi con cicli è possibile definire l'operazione di somma. Informalmente, dati due giochi imparziali $G_1 =< S_1, A_1, s_1>$ e $G_1 =< S_2, A_2, s_2>$, la somma $G_1 + G_2$ è il gioco in cui il giocatore di turno sceglie liberamente un gioco tra G_1 e G_2 e muove con le regole del gioco scelto, lasciando inalterato lo stato dell'altro. Il gioco termina quando non è possibile nessuna mossa in G_1 o in G_2 .

Come nel caso aciclico, la funzione di Grundy di una somma di giochi si calcola facilmente una volta nota la funzione di Grundy dei singoli giochi, e questo porta ad un algoritmo polinomiale per la determinazione di una strategia vincente per una somma di giochi, espressa in forma compressa A questo riguardo, ricordiamo che il valore $\gamma(v)$ della funzione di Grundy generalizzata o è un intero $a \in N$ oppure è una coppia $\infty(X)$, con X sottoinsieme finito di N. L'operazione "somma Nim" $\oplus$ può essere estesa alla "somma di Nim generalizzata" $\hat{\oplus}$ come segue:

1. $a \hat{\oplus} b = a \oplus b$ se $a,b \in N$
2. $\infty(X) \hat{\oplus} a = a \hat{\oplus} \infty(X) = \infty(X \oplus a)$, dove $X \oplus a = \{ x \oplus a \mid x \in X \}$
3. $\infty(X) \hat{\oplus} \infty(Y) = \infty(\varnothing)$

In perfetta analogia con la teoria sviluppata da Grundy e Sprague per giochi espressi da grafi aciclici, il seguente risultato esprime la funzione di Grundy di una somma di giochi in termini delle funzioni di Grundy generalizzate dei giochi presi separatamente.

Proposizione 2: $f_{G1+G2}(pq) = f_{G1}(p) \hat{\oplus} f_{G2}(q)$, dove pq è una coppia di vertici p e q dei grafi G_1 e G_2

5. Giochi in forma compressa

Abbiamo esibito una procedura che decide in tempo polinomiale l'esistenza di una strategia vincente nel caso di somme di giochi imparziali rappresentati attraverso grafi diretti (sia ciclici che aciclici). Discutiamo ora il caso generale di giochi (sia imparziali che partigiani) rappresentati in forma compressa.

In generale, un gioco in forma compressa può essere assegnato descrivendo in forma implicita le possibili configurazioni (stati) e le regole che permettono ad ogni giocatore di passare da una configurazione a una successiva con una mossa legale. Prenderemo qui in considerazione il caso in cui le possibili configurazioni sono infinite ma, fissata la configurazione iniziale, solo un numero finito di configurazioni è raggiungibile durante il gioco. Questo permette di istanziare una partita assegnando semplicemente lo stato iniziale, la cui dimensione è assunta come dimensione del gioco. Ne segue che:

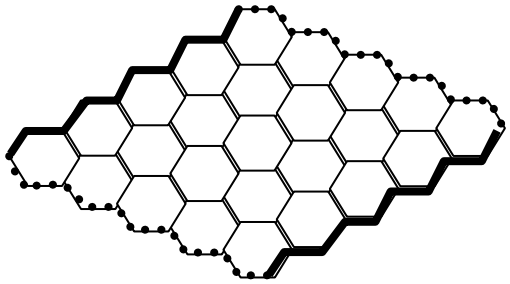
1. Un gioco in forma compressa riassume infiniti giochi, uno per ogni configurazione scelta come "stato iniziale"; la dimensione del singolo gioco è assunta essere quella dello stato iniziale.
2. Il numero di configurazioni raggiungibili dallo stato iniziale (e quindi la dimensione del grafo associato al singolo gioco) è generalmente un'esponenziale nella dimensione dello stato iniziale stesso (da cui il termine "gioco in forma compressa").

Esempio 1: il gioco *Node Kayles* è un gioco imparziale i cui stati sono grafi non orientati; dato uno stato, una mossa consiste nel rimuovere un vertice e i suoi adiacenti dal grafo. Il gioco inizia da un fissato grafo G=<V,E>, che possiamo pensare come "istanza" del gioco. Il grafo diretto associato al gioco ha come vertici i grafi ottenibili da G rimuovendo vertici e ha (G',G'') come arco se il grafo G'' è ottenuto rimuovendo un vertice e i suoi adiacenti da G'. Come si può osservare, tale grafo diretto è aciclico (ogni mossa produce un grafo con meno vertici, quindi non sono possibili cicli) che è usualmente di dimensione esponenziale rispetto a |V|, cioè alla dimensione dell'istanza del gioco. □

Esempio 2: il gioco *Annichilazione* è un gioco imparziale che viene giocato su una grafo diretto <V,A>, in cui inizialmente alcuni vertici contengono una marca. Il problema può essere descritto dal grafo con la marcatura iniziale. Una mossa consiste nel scegliere un vertice marcato e un arco uscente da esso, spostando la marca sul vertice di uscita; se sul vertice di uscita era tuttavia già presente una ulteriore marca, entrambe le marche vengono rimosse (*annichilite*). Anche in questo caso il grafo diretto associato al gioco può essere di dimensione esponenziale rispetto a |V|, cioè alla dimensione dell'istanza del gioco; tale grafo può tuttavia contenere cicli. □

Esempio 3: Il gioco *Ladro e k Poliziotti* viene giocato su un grafo non orientato G=<S,A>. Inizialmente sono poste sui vertici del grafo k marche nere (*i k poliziotti*) e una marca rossa (*il ladro*). A turno, il giocatore 1 (che controlla i k poliziotti) può spostare una marca nera a un vertice adiacente non occupato e il giocatore 2 (*il ladro*) la marca rossa a un vertice adiacente non occupato. Il giocatore 1 vince se riesce a forzare una configurazione in cui tutti vertici adiacenti a quello marcato in rosso sono marcati in nero (il ladro non ha scampo). □

Esempio 4: il gioco *Hex* è giocato su un grid esagonale 5x5 a forma di rombo, con coppie di lati opposti hanno lo stesso colore (rosso/nero). Il giocatore 1 ha a disposizione marche rosse, il 2 marche nere. La mossa di ogni giocatore consiste nel marcare col proprio colore un qualsiasi esagono vuoto: vince il giocatore che riesce a collegare con un cammino connesso monocolorato i due lati opposti.



Un grid esagonale 5x5

〈

Si osservi che il gioco Hex nella forma semplificata di sopra non può essere interpretato come *gioco in forma compressa*, poiché le possibili configurazioni sono finite. Sono tuttavia possibili varie generalizzazioni che permettono di vedere Hex come gioco in forma compressa. Una immediata generalizzazione è quella di considerare rombi nxn, per ogni n. Una seconda generalizzazione del gioco Hex a grafi ha come istanza un grafo non orientato G=<V,E>, in cui due vertici sono colorati in nero e ogni vertice ha al più grado 6. La mossa del giocatore 1 è di marcare in nero un vertice non ancora marcato, quella del giocatore 2 di marcare in rosso un vertice non ancora marcato. Il giocatore 1 vince se riesce a connettere i due vertici inizialmente colorati con un cammino marcato in nero. □

Abbiamo visto che un gioco in forma compressa riassume in realtà infiniti giochi, uno per ogni stato iniziale; ognuno di questi giochi ha una propria dimensione, che è assunta essere quella dello stato iniziale. E' pertanto possibile analizzare vari problemi che sorgono sui giochi in forma compressa utilizzando metodi di complessità asintotica. L'importante problema che analizziamo in dettaglio in queste note è quello dell'esistenza di strategie vincenti; fissato un gioco X in forma compressa, esso può essere formulato come il seguente problema di decisione:

Problema: *Strategia vincente in X*
Istanza: una configurazione s_0 del gioco X (stato iniziale)
Questione: se il giocatore 1 muove da s_0, ha una strategia vincente?

Il problema di decidere l'esistenza di strategie vincenti per giochi in forma compressa può allora essere affrontato con le classiche tecniche di complessità strutturale, basate sul concetto di classi di complessità e di problemi completi. A questo riguardo, richiamiamo quelle classi di complessità strutturale (e relativi problemi completi) che sono particolarmente utili alla classificazione del problema di decidere l'esistenza di strategie vincenti in giochi compressi.

6. Classi di complessità deterministiche, non deterministiche, alternanti.

Un problema di decisione viene descritto assegnando una questione (che ha come risposta "vero" o "falso") su varie istanze. Codificando opportunamente le istanze con parole su un dato alfabeto, il problema viene descritto estensivamente dal linguaggio formato dalle parole che codificano istanze che rendono vera la questione. *Risolvere* il problema significa allora *riconoscere* il linguaggio con un opportuno *algoritmo*, che nella sua esecuzione utilizzerà una certa quantità *di risorse*. *Classificare* un problema, in questo contesto, significa decidere se può essere risolto entro una fissata quantità di risorse o, equivalentemente, decidere se appartiene alla classe di linguaggi che possono essere riconosciuti entro un fissato limite di risorse (*classe di complessità*). Sviluppiamo qui le nozioni di base di complessità utlili alla classificazione del problema di decidere l'esistenza di strategie vincenti in giochi in forma compressa.

Un semplice e trasparente formalismo per il riconoscimento di linguaggi è la Macchina di Turing, di cui presentiamo qui informalmente un modello.

La macchina M accede con una testina a un nastro potenzialmente infinito, composto da celle in cui è possibile leggere e scrivere simboli da un insieme prefissato. Il *programma* della macchina consiste in un insieme finito di *istruzioni*, etichettate con *stati di controllo*. Per eseguire una *istruzione* etichettata con lo stato q la macchina: legge il simbolo nella cella individuata dalla testina, stampa un nuovo simbolo, muove eventualmente la testina (a destra o a sinistra), passa il controllo a una nuova istruzione p. Supponiamo inoltre che:

1. La macchina parte dallo stato q_0 , con la parola di ingresso w scritta sul nastro a partire dalla posizione 1 e con la testina in posizione 1; tutte le altre celle contengono il simbolo B.
2. La macchina "riconosce" la parola w se entra in un particolare stato q_{si} con la testina in posizione1, restando per i tempi successivi in tale configurazione, non la riconosce se entra in un particolare stato q_{no}.

Indichiamo con $\sum$ l'unione degli stati di controllo con le lettere dell'alfabeto di lavoro, simbolo B compreso. Con la parola xqy descriviamo la configurazione della macchina che si trova nello stato q, con la parola xy scritta sul nastro (escludendo i simboli B) e con la testina in posizione |x|+1. Data una configurazione xqy, la configurazione successiva può essere calcolata localmente, potendosi variare solo lo stato q e/o i simboli immediatamente a destra e a sinistra di q. In particolare, se $y_1.. y_n$ è la configurazione successiva a $x_1.. x_n$, il simbolo y_k dipende solo dai simboli $x_{k-2}x_{k-1}x_kx_{k+1}x_{k+2}$ (località del calcolo): porremo $y_k=g(x_{k-2}x_{k-1}x_kx_{k+1}x_{k+2})$, per una opportuna funzione $g:\sum^5\rightarrow\sum$, parziale nel senso che non è definita per tutti i valori di $\sum^5$. La macchina parte dalla configurazione q_0w e genera una sequenza di configurazioni C_1 …. C_k … C_t ,finchè entra in una delle configurazioni con stato q_{si} (e riconosce) o q_{no} (e non riconosce). Rimarchiamo che, fissata la configurazione iniziale $C_1= q_0w$, la sequenza di configurazioni generata è unica (modello deterministico) ed è calcolabile localmente!. Il numero t di tali configurazioni è il *tempo di calcolo* $T_M(w)$, la lunghezza della più lunga configurazione lo *spazio di calcolo* $S_M(w)$. Date due funzioni $f,g:N\rightarrow N$, M riconosce il linguaggio $L_M = \{w \mid w \text{ riconosciuto da M}\}$ in tempo f(n) (in spazio g(n)) se $Max_{|w|=n}\ T_M(w) \leq f(n)$ ($Max_{|w|=n}\ S_M(w) \leq g(n)$).

Obbiettivo della complessità strutturale è di classificare i problemi in classi di complessità. Le classi base in tempo o spazio deterministici sono:

1. TIME(f(n)) = { L | M riconosce L in tempo f(n) }
2. SPACE(g(n)) = { L | M riconosce L in spazio g(n) }

Classi di complessità (rispetto a modelli di calcolo deterministici) di interesse nell'area della teoria algoritmica dei giochi sono:

1. $P = \cup_k TIME(n^k)$
2. $PSPACE = \cup_k SPACE(n^k)$
3. $EXPTIME = \cup_k TIME(2^{\wedge} n^k)$

Il modello precedentemente presentato è deterministico. Esso può essere esteso considerando esplicitamente il *non determinismo*. In questo caso la macchina *non deterministica* M, fissata la configurazione iniziale q_0w, può generare più sequenze di calcolo: senza perdita di generalità, ipotizziamo che ogni configurazione ammetta al più 2 configurazioni successive; similmente al caso deterministico, le due eventuali configurazioni successive sono ottenibili localmente mediante due funzioni parziali $g,h:\Sigma^5 \rightarrow \Sigma$. La parola w di lunghezza n viene riconosciuta dalla macchina M in tempo f(n) (spazio g(n)) se almeno una sequenza di calcolo termina entrando nello stato q_{si} entro f(n) passi di calcolo (se la lunghezza di ogni configurazione della sequenza di calcolo è al più g(n)).

Analogamente al caso deterministico, si possono introdurre le classi basi di problemi risolubili entro un dato tempo o spazio:

1. $NTIME(f(n)) = \{ L \mid M$ non deterministica riconosce L in tempo $f(n) \}$
2. $NSPACE(g(n)) = \{ L \mid M$ non deterministica riconosce L in spazio $g(n) \}$

Di particolare rilievo sono le classi:

1. $NP = \cup_k NTIME(n^k)$
2. $NPSPACE = \cup_k SPACE(n^k)$

Oltre al caso deterministico e non deterministico, una ulteriore estensione che considereremo è il modello di macchina *alternante*. Una macchina alternante è una Macchina di Turing M non deterministica in cui gli stati sono partizionati in due classi: stati universali e stati esistenziali. Senza perdita di generalità, possiamo supporre che in ogni computazione della macchina gli stati esistenziali si alternano con stati universali e che lo stato iniziale si esistenziale.

Viene definito induttivamente il concetto di *configurazione accettante*.

1. La configurazione finale q_{si} è accettante.
2. Se q è uno stato esistenziale, la configurazione xqy è accettante se esiste almeno una configurazione accettante C ottenibile in un passo da xqy.
3. Se q è uno stato universale, la configurazione xqy è accettante se sono accettanti tutte le configurazioni C, ottenibili in un passo da xqy.

La parola w viene riconosciuta dalla macchina M se la configurazione iniziale q_0w è accettante.

Il concetto di computazione in una macchina alternante è immediatamente riferibile a quello strategia vincente in giochi. A questo riguardo, data una macchina alternante M, si consideri il gioco G(M) in cui:

- Gli *stati del gioco* sono le configurazioni della macchina
- (C,C') è *un arco del gioco* se C' è ottenibile da C in un passo della macchina M
- Nella configurazione accettante q_{si} vince il giocatore 1

E' immediato osservare che la parola w viene riconosciuta da M se e solo se il giocatore 1 ha una strategia vincente nel gioco G(M) partente dallo stato iniziale q_0w.

Continuando nell'analogia, se w è riconosciuta da M, il tempo di calcolo $T_M(w)$ della macchina M su ingresso w è la massima durata di una partita quando il giocatore 1 muove seguendo la strategia vincente ottima, mentre lo spazio di calcolo $S_M(w)$ è la lunghezza della più lunga configurazione che entra in una partita giocata dal giocatore 1 con la strategia vincente ottima. In seguito prenderemo in considerazione le seguenti classi di complessità:

1. ATIME(poly) ≡ classe di problemi risolubili in tempo polinomiale con macchine alternanti
2. ASPACE(poly) ≡ classe di problemi risolubili in spazio polinomiale con macchine alternanti

Per quanto visto sopra, ATIME(poly) formalizza la classe di giochi di *durata polinomiale* (rispetto all'istanza iniziale), mentre ASPACE(poly) la classe di giochi i cui stati raggiungibili sono di dimensione polinomiale (rispetto all'istanza iniziale).

Evidenziamo ora le principali relazioni tra le classi di complessità che abbiamo considerato. Evidentemente $P \subseteq NP \subseteq PSPACE \subseteq NPSPACE$. Mentre decidere se P=NP o $P \subset NP$ è il principale problema dell'informatica teorica ed è ancora aperto, i seguenti risultati sono stati provati da Savitch nel 1970 (W.J.Savitch, "Relationship between nondeterministic and deterministic tape classes", J.CSS, 4, 177-192, 1970) e da Chandra et al. nel 1981 (Chandra, A., Kozen, D., Stockmeyer, L., "Alternation", Jour. ACM, 114-133, 1981):

Fatto: NSPACE = PSPACE=ATIME(poly) , EXPTIME=ASPACE(poly)

Dimostrazione:

1. ***NSPACE $\subseteq$ ATIME(poly)***

Sia fissata una macchina non deterministica M che lavora in spazio polinomiale p(n) e quindi in tempo al più $(k+1)^{p(n)}$, dove k è la cardinalità dell'alfabeto. Siano A,B due configurazioni della macchina (quindi stringhe A,B con $|A|,|B| \leq p(n)$); con la notazione $(a,A) \rightarrow (b;B)$ intendiamo che c'è una sequenza di calcolo della macchina che al passo a si trova nella configurazione A e al passo b si trova nella configurazione B. Fissata una parola w con |w|=N, consideriamo il seguente gioco:

Gioco: *Savitch*
Stati del gioco: $\langle (a,A),(b,B) \rangle$, dove: A,B configurazioni di M, a,b interi $(1 \leq a < b \leq (k+1)^{p(n)})$.
Mosse dei giocatori (a partire dallo stato $\langle (a,A),(b,B) \rangle$):
giocatore1: se b=a+1, (giocatore 1) vince se $(a,A) \rightarrow (b,B)$ altrimenti perde
se b>a+1 allora (giocatore 1) calcola $c = \lceil (a+b)/2 \rceil$ e sceglie una configurazione C
giocatore 2: va nello stato $\langle (a,A),(c,C) \rangle$ oppure nello stato $\langle (c,C),(b,B) \rangle$

Lo *stato iniziale* del gioco è $\langle (1, q_0w), ((k+1)^{p(n)},C_{si}) \rangle$, dove C_{si} è la configurazione finale accettante.

Il giocatore 1 partendo da $\langle (1, q_0w), ((k+1)^{p(n)},C_{si}) \rangle$ ha una strategia vincente se e solo se w è accettato dalla macchina M. Infatti:

1. supponiamo che w sia accettato dalla macchina M, e quindi esista una sequenza di calcolo $C_1 = q_0w$, C_2 ,…, C_t, …, C_{si} della macchina M. Se ad ogni passo giocatore 1, dopo aver calcolato $c=\lceil (a+b)/2 \rceil$, sceglie la configurazione C_c, vince qualsiasi siano le scelte di giocatore 2, poiché se $\langle (a,A),(b,B) \rangle$ è uno stato raggiunto nel corso della partita, allora varrà sempre che $(a,A) \rightarrow (b,B)$.

2. supponiamo che w *non* sia accettato dalla macchina M. Questo significa che *è falso* che (1, q_0w) $\rightarrow$ $((k+1)^{p(n)},C_{si})$; se supponiamo di essere in uno stato <(a,A),(b,B)> per cui *è falso* che (a,A)$\rightarrow$(b,B), per qualsiasi scelta di C almeno una delle due seguenti affermazioni risulta *falsa*: (a,A)$\rightarrow$(c,C), (c,C)$\rightarrow$(b,B). Se il giocatore 2 muove sempre negli stati corrispondenti alle affermazione false allora vince.

L'esistenza di una strategia vincente del gioco *Savitch* è calcolabile quindi con una macchina alternante M' in un numero di passi pari alla massima durata di una partita. Indichiamo con T tale durata. Osserviamo che ogni stato <(a,A),(b,B)> del gioco individua un intervallo I=b-a con le seguenti caratteristiche: inizialmente l'intervallo è pari a $(k+1)^{p(n)}$, ad ogni turno l'intervallo si dimezza e il gioco termina quando l'intervallo è 1. Quindi la durata T del gioco è tale che $(k+1)^{p(n)}/2^T=1$, cioè T = p(n).log(k+1). Il problema è dunque risolto con una macchina alternante che lavora in tempo polinomiale. Questo prova che NPSPACE $\subseteq$ ATIME(poly). □

2. ***ATIME(poly)*** $\subseteq$ ***PSPACE***

Consideriamo una macchina alternante M che lavora in tempo polinomiale p(n) riconoscendo L_M . Ogni configurazione C raggiunta dalla macchina M che ha in ingresso una parola w con |w| =n avrà allora lunghezza |C|$\leq$ p(n); ogni configurazione, inoltre, ha al più un numero costante c=O(1) di configurazioni raggiungibili in un passo: tali configurazioni sono ordinabili (ad esempio in ordine lessicografico) in modo tale che la struttura della computazione è, astrattamente, un albero ordinato che ha come nodi le configurazioni, come radice la configurazione iniziale ed è tale che i figli di una configurazione C sono le configurazioni raggiungibili in un passo da C. Alle foglie aggiungiamo un bit di informazione (1 se la foglia è una configurazione accettante, 0 altrimenti), mentre ai nodi interni C associamo variabili booleane $x_1 \ldots x_c$ tante quante i figli.

Questo albero può essere percorso "*in profondità*". Nell'attraversamento dell'albero, il nodo C memorizza quale figlio è correntemente investigato; quando C riceve dal figlio di posto k un bit b (1 se il figlio è una configurazione accettante, 0 altrimenti), pone x_k=b. Dopo aver ricevuto i bit da tutti i suoi figli, C calcola la loro congiunzione (se C è una configurazione universale) o la loro disgiunzione (se C è una configurazione esistenziale) e invia il risultato al padre.

La parola w è accettata da M se il bit calcolato dalla radice dell'albero con il precedente algoritmo è 1. Poiché l'albero ha profondità p(n), l'algoritmo può essere implementato mediante una pila che contiene al massimo p(n) configurazioni: poiché ogni configurazione occupa spazio p(n), l'occupazione di memoria complessiva è al più $p^2(n)$. Abbiamo quindi costruito un algoritmo deterministico che riconosce L_M in spazio polinomiale. Allora ATIME(poly) $\subseteq$ PSPACE.

Poiché abbiamo provato che NSPACE $\subseteq$ATIME(poly) $\subseteq$ PSPACE$\subseteq$ NPSPACE, segue che NSPACE = PSPACE=ATIME(poly).

EXPTIME=ASPACE(poly) può essere provato con tecniche simili. □

7. Problemi completi

Allo scopo di classificare la difficoltà computazionale del problema dell'esistenza di una strategia vincente in giochi in forma compressa, sia pur in modo asintotico, abbiamo preso in considerazione le seguenti classi di complessità:

3. P ≡ classe di problemi risolubili in tempo polinomiale con algoritmi deterministici
4. ATIME(poly) ≡ classe di problemi risolubili in tempo polinomiale con macchine alternanti
5. PSPACE ≡ classe di problemi risolubili in spazio polinomiale con algoritmi deterministici ≡ classe di problemi risolubili in spazio polinomiale con algoritmi non deterministici
6. ASPACE(poly) ≡ classe di problemi risolubili in spazio polinomiale con macchine alternanti
7. EXPTIME ≡ classe dei problemi risolubili in tempo esponenziale ($2\text{^}n^{O(1)}$) con macchine deterministiche.

Abbiamo inoltre rilevato che:

Fatto: $P \subset EXPTIME$, $P \subseteq PSPACE \equiv ATIME(poly) \subseteq EXPTIME \equiv ASPACE(poly)$

Classificare un problema significa inserirlo al giusto livello della gerarchia di classi di complessità che abbiamo esibito. Ciò richiede due passi principali:

1. Inserire il problema in una classe, esibendo un algoritmo che utilizza livello di risorse richiesto per la classe.
2. Mostrare che questo è il massimo che si può fare, provando che il problema non può trovarsi in sottoclassi della classe data.

Il punto 2. è chiaramente di grande difficoltà. Richiamiamo qui un metodo introdotto in ambito di ricorsività (prima) e complessità strutturale (poi), che fornisce qualche risposta (spesso riducendo il problema a congetture più semplici).

L'idea basilare è il concetto di *riduzione*, che qui limitiamo alla riduzione polinomiale molti-uno.

Definizione 1: dati due linguaggi $L_1, L_2 \subseteq \sum^*$, diremo che L_1 è riducibile a L_2 (scrivendo $L_1 \angle L_2$) se esiste una funzione $f: \sum^* \rightarrow \sum^*$ tale che:

1. $w \in L_1$ se e solo se $f(w) \in L_2$.
2. f è calcolabile in tempo polinomiale da una Macchina di Turing deterministica

Questa definizione è giustificata dalla seguente proprietà, di diretta dimostrazione, che asserisce che ogni problema riducibile a un problema "facile da risolvere" è a sua volta "facile da risolvere".

Proposizione 1: se $L_1 \angle L_2$ e $L_2 \in P$, allora $L_2 \in P$.

Data una classe di linguaggi ***C***, un problema L è *difficile* per la classe ***C*** se ogni problema X della classe è riducibile a L. Se ulteriormente tale problema L è anche in ***C*** , L sarà detto *completo*.
In sostanza, un problema completo nella classe ***C*** è un massimo della relazione di preordine $\angle$.

Se tale problema è "facile da risolvere" allora ogni problema della classe è "facile da risolvere" (cioè vale $\boldsymbol{C} \subseteq P$). Supponiamo che invece sia $P \subset \boldsymbol{C}$: in questo caso un problema ***C*** completo non potrà essere risolto in tempo polinomiale.

Per le classi NP, PSPACE e EXPTIME sono stati esibiti problemi completi, alcuni dei quali discussi in seguito. Poiché $P \subset EXPTIME$, un problema EXPTIME completo non può essere risolto in tempo polinomiale. Non si conosce invece se $P \subset NP$ e nemmeno se $P \subset PSPACE$, anche se la cosa è ritenuta verosimile dai più. Accettando tali congetture, possiamo quindi asserire che problemi NP completi e PSPACE completi non sono in P.

Una tecnica per provare la PSPACE (EXPTIME, NP) completezza di un problema Y è quella di ridurre in tempo polinomiale un problema completo X a Y, provando poi che Y è in PSPACE (EXPTIME, NP).

Questa tecnica richiede tuttavia di conoscere alcuni problemi completi. Il primo problema NP completo, trovato indipendentemente da Cook e Levin nel 1970, è il classico problema di soddisfacibilità nella logica enunciativa, che andiamo brevemente a descrivere.

Consideriamo variabili booleane $x_1,x_2, x_3 \ldots , x_n , \ldots$Ad ognuna di tali variabili può essere assegnato il valore 0 oppure 1. Dati i connettivi logici $\wedge,\vee,\neg$, il concetto di *formula* sulle variabili $x_1,x_2, \ldots , x_n$ viene definito induttivamente:

1. Le variabili $x_1,x_2, \ldots , x_n$ sono *formule*
2. Se Φ e Ψ sono *formule*, allora $\Psi \wedge \Phi$, $\Psi \vee \Phi$ e $\neg\Phi$ sono *formule*

L'implicazione $\Phi \Rightarrow \Psi$ è una abbreviazione di $\neg\Phi \vee \Psi$, lo aut (or esclusivo) $\Phi \oplus \Psi$ è un'abbreviazione di $(\Phi \vee \Psi) \wedge (\neg\Phi \vee \neg\Psi)$. Dato un assegnamento $a_1,a_2, \ldots , a_n$ alle variabili, $\Psi \wedge \Phi$ vale 1 se sia Ψ che Φ valgono 1, $\Psi \vee \Phi$ vale 0 se sia Ψ che Φ valgono 0, $\neg\Phi$ vale 1 se Φ vale 0. Formule corrispondenti a variabili x_k o variabili negate $\neg x_k$ sono dette *letterali*; congiunzioni ($\wedge$) di letterali sono dette *monomi*, disgiunzioni ($\vee$)di letterali sono dette clausole; una *forma congiunta* è una congiunzione di clausole, mentre una *forma disgiunta* è una disgiunzione di monomi.
Il problema di soddisfacibilità SAT (Satisfiability) è il seguente:

Problema: SAT
Istanza: una formula booleana $\Psi(x_1,x_2, .. , x_n)$
Questione: E' tale formula sodddisfacibile ? (esiste un assegnamento $a_1,..,a_n$ per cui $\Psi(a_1,..,a_n)=1$)

Sat è un classico problema NP completo, come asserito in:

Proposizione 2: SAT è NP-completo.
Dimostrazione:

SAT è in NP.

Un algoritmo per SAT consiste nel costruire in modo non deterministico un assegnamento $a_1,..,a_n$ (tempo non deterministico lineare) e nel verificare se $\Psi(a_1,..,a_n)=1$ (tempo deterministico polinomiale).

SAT è NP-difficile.

Bisogna provare che, preso un arbitrario linguaggio L in NP, $L \angle SAT$. Poiché L è in NP, esiste una macchina non deterministica M che riconosce L in tempo polinomiale p(n). Basta associare, in tempo polinomiale, a una parola w una formula Ψ_w tale che $w \in L$ se e solo se Ψ_w è soddisfacibile. Osserviamo che $w \in L$ se e solo se:

esiste una sequenza di configurazioni $C_1 \ldots C_{p(n)}$, dove $n=|w|$, tale che:

$$(*) \qquad in(C_1) \wedge \bigwedge_{1 \le t \le p(n)-1} succ(C_t, C_{t+1}) \wedge fin(C_{p(n)})$$

dove:

1. in(C)=1 se $C=q_0w$ (in(-) è una relazione che attesta che C è la configurazione iniziale)
2. fin(C)=1 se $C= q_{si}$ (fin(-) è una relazione che attesta che C è la configurazione accettante)
3. succ(X,Y)=1 se la configurazione Y è un possibile successore della configurazione X.

Per descrivere con una formula booleana la generica configurazione C utilizzeremo variabili booleane $S(p,\sigma)$ col significato:

$S(p,\sigma)=1 \approx$ nella posizione p della configurazione C c'è il simbolo σ

Nello stesso modo, per descrivere con una formula booleana una sequenza di configurazioni $C_1 \ldots C_{p(n)}$ generata dalla macchina utilizzeremo variabili booleane $S(t,p,\sigma)$ col significato:

$S(t,p,\sigma)=1 \approx$ nella posizione p della configurazione C_t al passo t c'è il simbolo σ

Di conseguenza:

1. $In(C) \equiv S(1,0, q_0) \wedge S(1,p, w_1) \wedge S(1,p, w_2) \wedge \ldots S(1,p, w_n)$, dove $w= w_1 w_2 \ldots w_n$.
2. $fin(C) \equiv S(p(n),0, q_{si})$
3. Per descrivere succ(X,Y), supponiamo che $X= x_0 x_1 \ldots x_n$ e $Y= y_0 y_1 \ldots y_n$. Ricordiamo che la macchina ha al più due configurazioni successive a X, calcolabili localmente. Data una configurazione X e un suo fattore $x_{k-2} x_{k-1} x_k x_{k+1} x_{k+2}$, sono possibili al più due valori di y_k che vanno a formare le due possibili configurazioni successive. Poniamo tali valori uguali a $g(x_{k-2} x_{k-1} x_k x_{k+1} x_{k+2})$ e $h(x_{k-2} x_{k-1} x_k x_{k+1} x_{k+2})$ (se un solo valore è possibile poniamo $h(x_{k-2} x_{k-1} x_k x_{k+1} x_{k+2})$ indefinito). Allora:

$$succ(X,Y) \equiv$$

$$\equiv \bigwedge_k [S(k-2, x_{k-2}) \wedge \ldots \wedge S(k+2, x_{k+2})] \Rightarrow S(k,g(x_{k-2} x_{k-1} x_k x_{k+1} x_{k+2})) \oplus S(k,h(x_{k-2} x_{k-1} x_k x_{k+1} x_{k+2}))$$

Nell'espressione precedente, se $h(x_{k-2} x_{k-1} x_k x_{k+1} x_{k+2})$ non è definita conveniamo di sostituirla con 0; l'operazione $\oplus$ serve a imporre che le due scelte non deterministiche g e h sono mutuamente esclusive.

La espressione (*) può allora essere sostituita da una formula di lunghezza $O(p(n)^2)$; tale formula è soddisfacibile se e solo se la parola $w \in L$. □

Una analisi più dettagliata della formula ricavata nella precedenti dimostrazione mostra che essa è essenzialmente una forma congiunta con clausole di al più 7 letterali. Con facili sostituzioni, inoltre, si può provare che è NP completo il problema di soddisfacibilità per forme congiunte con clausole di al più 3 letterali.

Il classico problema PSPACE completo è TQBS (True Quantified Boolean Formulas), immediatamente riferibile all'esistenza di strategie vincenti:

Problema: TQBF
Istanza: una formula booleana con quantificatori alternati $\exists x_1 \forall x_2 \exists x_3 \ldots \exists x_n \Psi(x_1,x_2, .. , x_n)$
Questione: E' tale formula quantificata vera?

Proposizione 3: TQBF è PSPACE-completo.
Dimostrazione:

TQBF è in PSPACE

E' immediato osservare che TQBF può essere risolto in tempo lineare con una Macchina di Turing alternante. Allora è in PSPACE ricordando che PSPACE=ATIME(poly)

TQBF è PSPACE difficile

Sia L un linguaggio in PSPACE. Bisogna provare che, preso un arbitrario linguaggio L in PSPACE, $L \angle SAT$. Poiché L è in PSPACE, esiste una macchina alternante M che riconosce L in tempo polinomiale p(n).

Basta associare, in tempo polinomiale, a una parola w una formula $\exists x_1 \forall x_2 \ldots \exists x_N \Psi(x_1, x_2, .., x_N)$ tale che $w \in L$ se e solo se la sentenza $\exists x_1 \forall x_2 \ldots \exists x_N \Psi(x_1, x_2, .., x_N)$ è vera.

Osserviamo che $w \in L$ se e solo se:

esiste una conf. C_1 per ogni conf. C_2... esiste una con. $C_{p(n)}$, dove $n=|w|$, tale che:

$$(*) \qquad in(C_1) \wedge \bigwedge_{1 \le t \le p(n)-1} succ(C_t, C_{t+1}) \wedge fin(C_{p(n)})$$

dove:

4. in(C)=1 se $C=q_0 w$ (in(-) è una relazione che attesta che C è la configurazione iniziale)
5. fin(C)=1 se $C= q_{si}$ (fin(-) è una relazione che attesta che C è la configurazione accettante)
6. succ(X,Y)=1 se la configurazione Y è un possibile successore della configurazione X.

La sequenza di configurazione $C_1 \ldots C_{p(n)}$ generata dalla macchina può infine essere descritta da assegnamenti verificanti una formula booleana esattamente come descritto in **Proposizione 2.**

Un classico problema EXPTIME completo è infine il problema dell'ARRESTO LIMITATO, che richiede di decidere se una data Macchina di Turing M su un dato ingresso termina x il calcolo entro un dato tempo T. Equivalentemente, data una Macchina di Turing universale U sull'alfabeto Σ, tale problema può essere così formulato:

Problema: ARRESTO LIMITATO
Istanza: una parola $w \in \Sigma$
Questione: su ingresso w, la macchina U si arresta in $2^{|w|}$ passi?

Come abbiamo anticipato, possiamo provare:

Proposizione 4: ARRESTO LIMITATO è EXPTIME-completo
Dimostrazione:
Per mostrare che ARRESTO LIMITATO è EXPTIME completo, si osservi intanto che tale problema è in EXPTIME, in quanto risolubile in tempo 2^n. Dato ora un qualsiasi linguaggio L in EXPTIME, quindi riconoscibile da una macchina M in tempo $2^{p(n)}$ per un certo polinomio p, mostriamo che L è riducibile a ARRESTO LIMITATO. Possiamo per prima cosa considerare la macchina M' che accetta y se e solo se y è della forma $y= wa^{2p(|w|)}$ con w accettato da M ed a è un simbolo non in w; poiché $|y|=|w|+2.p(|w|)$, M' lavora quindi in tempo $2^{|y|}$.

La riduzione si ottiene associando alla parola w la parola z(w) che codifica $<M', wa^{2p(|w|)}>$. Vale:

1. La funzione che associa w a z(w) è chiaramente calcolabile in tempo polinomiale.
2. La macchina U, ricevendo in ingresso $z(w)= <M', wa^{2p(|w|)}>$, simula M' su $wa^{2p(|w|)}$, lavorando in tempo $2^{|z(w)|}$.
   Ne segue che $w \in L$ se e solo se $<M', wa^{2p(|w|)}> \in$ ARRESTO LIMITATO.

8. Giochi in PSPACE e EXPTIME

In questa Sezione osserviamo che per una notevole sottoclasse di giochi combinatori (anche partigiani) il problema di decidere l'esistenza di una strategia vincente è in PSPACE, o almeno in EXPTIME. Molti giochi in forma compressa verificano infatti le ipotesi descritte nelle seguenti proposizioni:

Proposizione 1: se ogni configurazione raggiungibile dallo stato iniziale di un gioco Y è descrivibile in spazio polinomiale rispetto alla dimensione dello stato iniziale e le mosse legali possono essere riconosciute in tempo polinomiale, allora l'esistenza di una strategia vincente del giocatore 1 è un problema in EXPTIME.

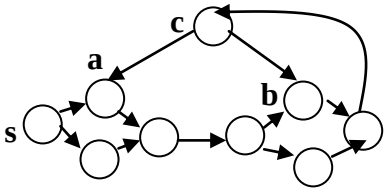
Dimostrazione: il problema di esistenza di una strategia vincente per Y può essere risolto attraverso una macchina alternante che lavora in spazio polinomiale rispetto alla dimensione dell'istanza.
Tale problema risulta quindi in ASPACE(poly) e di conseguenza in EXPTIME, poiché EXPTIME≡ASPACE(poly).

Proposizione 2: se ogni configurazione raggiungibile dallo stato iniziale di un gioco Y è descrivibile in spazio polinomiale, le mosse legali possono essere riconosciute in tempo polinomiale e il numero massimo di passi di una partita è limitato da un polinomio nella dimensione dello stato iniziale, allora l'esistenza di una strategia vincente del giocatore 1 è un problema in PSPACE.

Dimostrazione: nelle ipotesi date il problema è risolto da una macchina alternante che lavora in tempo polinomiale, quindi il problema di esistenza di una strategia vincente per Y in PSPACE, poiché ATIME(poly)=PSPACE.

Esempio 1: Consideriamo il gioco *Geografia*. E' assegnato un grafo orientato G=<V,A> e un vertice s iniziale contenente una marca. I giocatori muovono a turno, spostando la marca da un vertice al successivo lungo un arco, eliminando poi dal grafo l'arco. Vince il giocatore che forza l'altro a non avere più mosse legali.

Nel gioco qui a destra, se il gioco parte da s e la partita passa per **a** e **b**, allora vince il giocatore 1 (il giocatore 2 non può muovere da **a** o **b** poiché gli archi entranti o uscenti sono rimossi). In ogni altro caso vince il giocatore 2.



Poiché ad ogni mossa viene rimosso un vertice, il numero di mosse in una partita è al massimo |V|, lineare quindi nella dimensione dell'istanza G=<V,A>. Il problema di decidere l'esistenza di strategie vincenti in *Geografia* è allora in PSPACE □

Esempio 2: consideriamo la generalizzazione del gioco Hex (vedi **Esempio 4** di Sez.) in cui lo stato iniziale è un grafo non orientato G=<V,E> con vertici di grado al più grado 6 e con due vertici colorati in nero. Alternativamente il giocatore 1 e il giocatore 2 marcano rispettivamente in nero (in rosso) un vertice non ancora marcato; obbiettivo del giocatore 1 è di creare un cammino marcato in nero tra i due vertici segnati in partenza. In questo gioco, in ogni mossa viene colorato un vertice: il numero di vertici è allora un limite massimo al numero di mosse per partita, quindi il problema di determinare una strategia vincente è in PSPACE. □

Esempio 3: il gioco *Annichilazione* (vedi **Esempio 2** di Sez.) è un gioco imparziale che viene giocato su una grafo diretto <V,A>, in cui inizialmente alcuni vertici contengono una marca. I giocatori scelgono alternativamente un vertice marcato e spostano la marca su un vertice di uscita; se su tale vertice era già presente una marca, entrambe le marche vengono rimosse. Il vincitore è determinato quando tutte le marche sono rimosse.

Poiché ogni stato del gioco è una marcatura di <V,A>, esso è descrivibile in spazio lineare rispetto all'istanza e le mosse legali possono essere riconosciute in tempo polinomiale (quindi l'esistenza di una strategia vincente è un problema in EXPTIME).; non è invece possibile dare un limite polinomiale alla durata di ogni partita, poiché il grafo associato al gioco contiene cicli, e quindi apparentemente non possiamo classificare il problema in PSPACE. Ciò nonostante, una analisi dettagliata prova che il problema dell'esistenza di strategie vincenti può essere risolto in tempo n^6 , dove n è il numero dei vertici del grafo. (A. S. Fraenkel, Y. Yesha, Theory of annihilation games--I, J. Combin. Theory Ser. B 33, 60-86 (1982). □

Esempio 4: Analogamente al caso precedente, anche nel gioco *Ladro e k Poliziotti* (vedi **Esempio 3** in Sez.) ogni configurazione del gioco raggiungibile da quella iniziale è descrivibile in spazio lineare rispetto dimensione dello stato iniziale e le mosse legali possono essere riconosciute in tempo polinomiale; possiamo concludere che il problema di decidere strategie vincenti è in EXPTIME. Apparentemente non possiamo classificare il problema in PSPACE, poichè la lunghezza di una partita non è generalmente limitata polinomialmente nel numero di vertici del grafo. Questa intuizione può essere provata in modo rigoroso, dimostrando che tale problema è EXPTIME completo (Goldstein,A.S., Reingold, E.M., The complexity of pursuit on a graph, Theor.Comp.Sci., 143:93-112, 1995). □

Esempio 5: Una "versione partigiana" di *Annichilazione* è il gioco *Cattura.* In tale gioco le marche possono avere due colori (rosso/nero). Il giocatore 1 muove le marche rosse, il giocatore 2 le marche nere. Ogni marca può essere mossa in una posizione non contenente una marca dello stesso colore; se una marca viene mossa in un vertice contenente una marca di colore opposto, questa seconda marca viene rimossa (catturata). Se il grafo su cui si gioca *Cattura* è aciclico, allora il durata di una partita è polinomiale nel numero di nodi. La determinazione di una strategia vincente è allora un problema in PSPACE per giochi su grafi aciclici, mentre per giochi generali è un problema in EXPTIME.

9. Giochi PSPACE completi

Abbiamo visto che l'esistenza di strategie vincenti per vari giochi è in PSPACE o in EXPTIME. Questo non dice nulla sulla possibilità o meno che problema sia "trattabile" algoritmicamente, cioè risolubile in tempo polinomiale. Ad esempio, il problema *Annichilazione*, classificato subito in EXPTIME e di cui non era immediatamente evidente la appartenenza a PSPACE, ammette un (non banale) algoritmo risolutivo in tempo polinomiale.

E' quindi importante trovare argomenti che provano rigorosamente l'intrinseca difficoltà di risolvere il dato problema. Ad esempio, abbiamo osservato che il gioco *Geografia* (vedi **Esempio 1** di Sez.) è in PSPACE. E' questo il massimo che si riesce ad ottenere oppure, come nel caso del gioco *Annichilazione*, una più attenta analisi permette di determinare un algoritmo risolutivo polinomiale?

Mostriamo qui che non ci si può aspettare l'esistenza di algoritmi polinomiale per *Geografia*, provando che tale problema è PSPACE completo. L'esistenza di un algoritmo che lavora in tempo polinomiale per *Geografia* implicherebbe allora che P=PSPACE (contro la congettura generalmente accettata che P≠PSPACE). Questo risultato è dimostrato in (Schaefer, T.J., On the complexity of some two-person perfect information game, Jour.of Comp. and Syst.Sci., 16:185-225, 1978) con una tecnica prototipale utile a provare l'intrattabilità di numerosi problemi su giochi, a cui *Geografia* può essere ridotto.

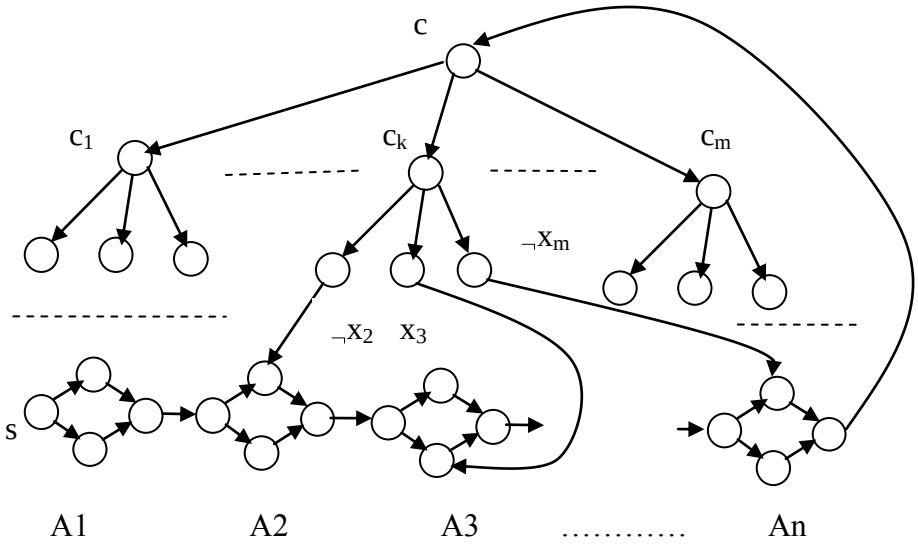
Proposizione 1: decidere l'esistenza di strategie vincenti per *Geografia* è un problema PSPACE completo.
Dimostrazione:come abbiamo visto, poiché ogni partita di Geografia ha una durata al più pari al numero di vertici del grafo istanza, il problema è in PSPACE. Il fatto che tale problema è PSPACE difficile può essere invece essere dimostrato riducendo ad esso il classico problema PSPACE completo TQBS (True Quantified Boolean Formulas):

Problema: TQBF
Istanza: una formula booleana con quantificatori alternati $\exists\, x_1 \forall x_2 \exists\, x_3 \ldots \exists\, x_n\, \Psi(x_1,x_2, .. , x_n)$
Questione: E' tale formula quantificata vera?

Questo problema resta completo anche se $\Psi(x_1,x_2, .. , x_n)$ è una congiunzione $c_1 \wedge .. \wedge c_m$ di clausole $c_k = e_k \vee f_k \vee g_k$ di 3 letterali ($1 \le k \le m$).

Ad ogni formula $\Psi(x_1,x_2, .. , x_n)$, in cui senza perdita di generalità possiamo supporre n dispari, associamo un grafo composto da (vedi disegno seguente):

1. n blocchi $A_1, \ldots, A_n$, di 4 nodi l'uno, collegati sequenzialmente uno all'altro; il primo nodo di A_1 è il vertice iniziale s; ogni blocco A_k ha un nodo "in alto" e uno "in basso" che individuano i due possibili cammini del blocco; l'ultimo nodo di A_n è collegato con un nodo c.
2. m nodi $c_1,.., c_m$, uno per ogni clausola di Ψ; c è collegato a ognuno di questi nodi
3. ogni nodo c_k è collegato con nodi etichettati dai letterali contenuti nella corrispondente clausola
4. ogni nodo etichettato da un letterale positivo x_j è collegato col nodo "in basso" di A_j, mentre ogni nodo etichettato da un letterale negativo $\neg x_j$ è collegato col nodo "in alto" di A_j



La seguente regola mette in corrispondenza biunivoca i cammini da s a c con gli assegnamenti alle variabili $x_1,x_2, .. , x_n$: al cammino C viene associato l'assegnamento $a_1,a_2, .. , a_n$, dove $a_k = 1$ se C passa per il vertice "in alto" di A_k mentre $a_k = 0$ se C passa per il vertice "in basso" di A_k $(1 \leq k \leq n)$.

Osserviamo che il giocatore 1 può vincere la partita iniziante col cammino C, dal vertice s a c, se e solo se $\Psi(a_1,a_2, .. , a_n)=1$.

Supponiamo infatti che $\Psi(a_1,a_2, .. , a_n)=1$ e che il giocatore 2 muova da c ad un qualsiasi vertice c_k. Poiché la clausola c_k sull'assegnamento $a_1,a_2, .. , a_n$ vale 1, almeno uno dei letterali che compongono c_k vale1. Se il giocatore 1 muove nel vertice etichettato con tale letterale allora vince (perché la partita è forzata a terminare dopo due mosse).

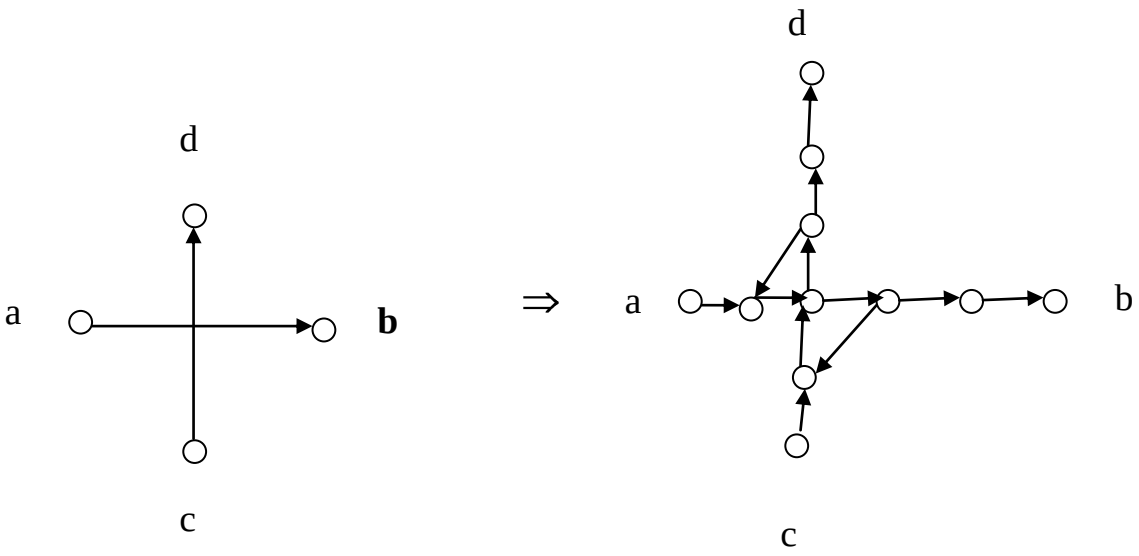
Supponiamo invece che $\Psi(a_1,a_2, .. , a_n)=0$ e che quindi esista una clausola c_k che sull'assegnamento $a_1,a_2, .. , a_n$ vale 0. Se il giocatore 2 da c muove in tale clausola, allora vince la partita perché, in qualsivoglia vertice etichettato con un letterale di c_k il giocatore 1 vada, la partita è forzata a terminare dopo la mossa seguente.

Allora il giocatore 1 ha una strategia vincente se e solo se $\exists\, x_1 \forall x_2 \exists\, x_3 \,\ldots\, \exists\, x_n\, \Psi(x_1,x_2, .. , x_n)$. □

L'istanza di un problema *Geografia* è un grafo orientato arbitrario. Se ci si limita a considerare grafi planari il problema viene denominato *Geografia Planare*. Anche limitato a tali grafi il problema rimane PSPACE completo:

Proposizione 2: decidere l'esistenza di strategie vincenti per *Geografia planare* è un problema PSPACE completo.
Dimostrazione: basta ridurre *Geografia* a *Geografia Planare*. A tal riguardo, dato un grafo diretto <V,A> (istanza di *Geografia*), associamo ad esso un grafo planare <V',A'> applicando il seguente algoritmo:

1. Disegna <V,A> su un piano
2. se due archi si intersecano, sostituiscili col grafo rappresentato nella figura seguente:



Partendo da <V,A>, costruiamo pertanto un nuovo grafo planare <A',V'>.
Supponiamo, senza perdita di generalità, che nel gioco su <V,A> l'arco **(a,b)** venga percorso prima di **(c,d).**

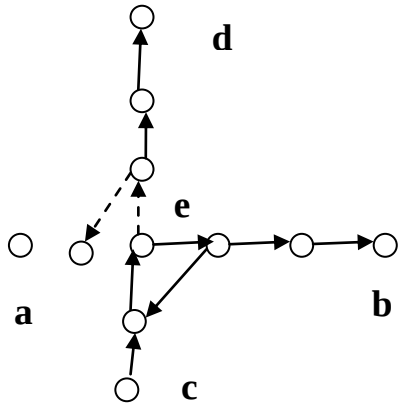


Fig.1

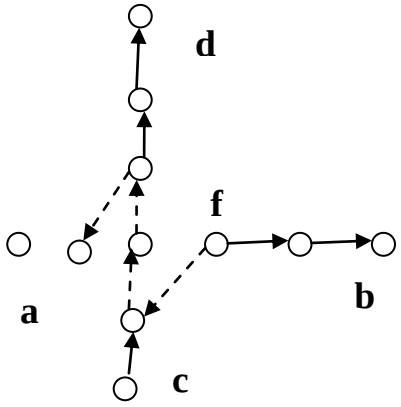


Fig.2

Se, nel corrispondente cammino nel grafo <A',V'>, il giocatore di turno, arrivato in **e**, non muove verso **b** (quindi muove in **d**) , l'avversario ha una strategia vincente (Fig.1). Lo stesso succede se il giocatore di turno da **f** non muove verso **b** (Fig.2). Quindi se un giocatore tenta di percorrere il cammino da **a** a **b** nel grafo <V',A'>, e nessuno dei due giocatori vuole perdere, dopo 5 passi l'altro si troverà in **b**.

Questo implica che il giocatore 1 ha una strategia vincente su <V,A> se e solo se il giocatore 1 ha una strategia vincente su <V',A'>, e quindi abbiamo costruito una riduzione da *Geografia* a *Geografia Planare* . □

10. Giochi EXPTIME completi

In questa sezione introduciamo un gioco EXPTIME completo.
Richiamiamo il classico problema EXPTIME completo "ARRESTO LIMITATO". Per specificare tale problema, si fissi una macchina di Turing universale U con alfabeto di ingresso $\{0,1\}$:

Problema ARRESTO LIMITATO:
Istanza: una parola $y \in \{0,1\}^*$
Questione: la macchina U accetta y e termina entro $2^{|y|}$ passi di calcolo.

Allo scopo di presentare un gioco EXPTIME completo, consideriamo una Macchina di Turing universale U a un nastro e con alfabeto di ingresso 0,1. Sia Q l'insieme degli stati, Γ l'alfabeto di lavoro contenente 0,1 e poniamo $\Sigma = .\Gamma \cup Q$. Supponiamo che:

3. La macchina parte dallo stato q_0 , con la parola di ingresso w scritta sul nastro a partire dalla posizione 1 e con la testina in posizione 1
4. La macchina "riconosce" la parola w se entra in un particolare stato q_{si} con la testina in posizione1, restando per i tempi successivi in tale configurazione.

Con la parola xqy descriviamo la configurazione della macchina che si trova nello stato q, con la parola xy scritta sul nastro e che la testina in posizione |x|+1; con $C_w(t,p)$ denotiamo il simbolo in posizione p nella configurazione raggiunta in t passi di calcolo da U, inizializzata con la parola w scritta sul nastro. La configurazione iniziale, trascurando i caratteri di blank a destra e sinistra, sarà allora $c = q_0w$. Data una configurazione $x\ A_{-2}\ A_{-1}\ A_0\ A_1\ A_2\ y$, la configurazione al passo successivo $x'\ A'_{-2}\ A'_{-1}\ A'_0\ A'_1\ A'_2\ y'$ è univocamente definita; si osserva inoltre che A'_0 dipende solo da $A_{-2}\ A_{-1}\ A_0\ A_1\ A_2$. Questo ci permette di definire la funzione (parziale) $\delta : \Sigma^5 \to \Sigma$, con $\delta(A_{-2}, A_{-1}, A_0, A_1, A_2) = A'_0$.

Per ogni parola $w \in \{0,1\}^*$, con ALT(w) intendiamo il seguente gioco:

Gioco ALT(w)

1. Gli stati del gioco sono le quadruple (w,p,t,A) oppure $(w,p,t, A_{-2}\ A_{-1}\ A_0\ A_1\ A_2)$, con $w \in \{0,1\}^*$, p e t interi positivi, le lettere A, $A_{-2}, A_{-1}, A_0, A_1, A_2$ appartenenti a Σ.
2. Lo stato iniziale è $(w,1, 2^{|w|}, q_{si})$
3. Il giocatore 1 parte da (w, p, t+1, A) muovendo in $(w, p, t, A_{-2}\ A_{-1}\ A_0\ A_1\ A_2)$, a patto che $A=\delta(A_{-2}, A_{-1}, A_0, A_1, A_2)$ e, se t=1, $A_s = c_{p+s}$ per $-2 \le s \le 2$ (ricordiamo che $c= q_0w$) .
4. Il giocatore 2 muove da $(w ,p ,t , A_{-2}A_{-1}A_0A_1A_2)$ solo se t>1; in tal caso prima sceglie $s \in \{-1,0,1\}$ poi va nello stato $(w,p+s,t, A_s)$

Supponiamo ora che la macchina U accetti w in al più $2^{|w|}$ passi.
In questo caso il giocatore 1 ha una strategia vincente ottenibile analizzando la sequenza di configurazioni generata U, inizializzata con la parola w.
Il giocatore 1 infatti:

1. parte da uno stato iniziale $(w,1, 2^{|w|}, q_{si})$ con $q_{si} = C_w(2^{|w|},1)$, poiché la parola w è accettata entro $2^{|w|}$ passi di calcolo
2. può forzare il gioco in modo da muovere sempre da stati (w,p,t,X) dove $X= C_w(t,p)$. Supponiamo infatti che il giocatore 1 parta da (w,p,t+1,A) con $A= C_w(t+1,p)$; poiché per definizione $C_w(t+1,p)= \delta(C_w(t,p-2), C_w(t,p-1), C_w(t,p), C_w(t,p+1), C_w(t,p+2))$, basta che il giocatore 1 muova a $(w,p,t, A_{-2}\ A_{-1}\ A_0\ A_1\ A_2)$ con $A_s = C_w(t,p+s)$ per $-2 \le s \le 2$; se t>1 il giocatore 2 non potrà che scegliere $s \in \{-1,0,1\}$ e muovere nello stato $(w,p+s,t, A_s)$ con $A_s = C_w(t,p+s)$, mentre se t=1 il giocatore 2 perde.

Supponiamo invece che la macchina U o non accetti w oppure accetti w in più di $2^{|w|}$ passi.
In questo caso il giocatore 2 può forzare il giocatore1 alla sconfitta. Infatti:

1. Il giocatore 1 parte da uno stato iniziale $(w,1, 2^{|w|}, q_{si})$ con $q_{si} \neq C_w(2^{|w|},1)$, poiché altrimenti la parola w sarebbe stata accettata entro $2^{|w|}$ passi di calcolo
2. Il giocatore 2 può forzare il giocatore 1 a muovere da stati (w,p,t, X) con $X \neq C_w(t,p)$. Supponiamo infatti che il giocatore 1 parta da (w,p,t+1,A) con $A \neq C_w(t+1,p)$ e che muova a $(w,p,t, A_{-2}\, A_{-1}\, A_0\, A_1\, A_2\,)$ con $A=\delta(\, A_{-2},A_{-1},A_0,A_1,A_2)$; esiste s $(-2 \leq s \leq 2)$ con $A_s \neq C_w(t,p+s)$ perché altrimenti $\delta(\, C_w(t,p-2), C_w(t,p-1), C_w(t,p), C_w(t,p+1), C_w(t,p+2)) = C_w(t+1,p)$, mentre sappiamo che $A \neq C_w(t+1,p)$. Basta che il giocatore 2 scelga tale valore di s ed 1 si trova nuovamente in uno stato $(w,p+s,t, A_s\,)$ con $A_s \neq C_w(t,p+s)$.
3. Se infine il giocatore 1 si trova nello stato (w,p,2,X) con $X \neq C_w(2,p)$, allora, per qualche s, dovrà essere $A_s \neq c_{p+s} = C_w(1,p+s)$ (altrimenti sarebbe $X = C_w(2,p)$); il giocatore 1 non può quindi muovere da (w,p,2,X) e perde.

La discussione precedente permette di concludere che il giocatore 1 ha una strategia vincente nel gioco Alt(w) se e solo se la macchina di Turing universale U accetta w entro $2^{|w|}$ passi di calcolo. Ma questo significa che "Bounded halting" è riducibile al problema di decidere l'esistenza di una strategia vincente per Alt(-) e quindi vale:

Proposizione 1: decidere l'esistenza di strategie vincenti nel gioco Alt(-) è un problema EXPTIME completo.

11. Giochi “contro Natura” e dintorni: la classe IP

Il modello di gioco che abbiamo considerato fino ad ora non prevede mosse probabilistiche: il giocatore che effettua una mossa fa una scelta libera (non deterministica) all’interno delle possibilità legali. Questo crea una certa similitudine di comportamento tra i giocatori, ognuno dei quali cerca di individuare una strategia vincente. Il modello ideale è la Macchina alternante. Fissato l’ingresso w (inizio del gioco) il giocatore 1 muove da configurazioni esistenziali con l’obbiettivo di arrivare in una configurazione accettante (cerca di accettare la parola w) mentre il giocatore 2 muove da configurazioni universali e cerca di arrivare in una configurazione non accettante (cerca di rifiutare la parola w): la parola w viene accettata quando il giocatore 1 ha una strategia vincente.

Possiamo modificare il modello imponendo al giocatore 2 di muovere in modo probabilistico: partendo da una configurazione C, il giocatore 2 prima genera una stringa B di bit casuali, poi applica un algoritmo deterministico a <C,B> ottenendo una configurazione C’ e passando il controllo al giocatore 1. Il giocatore 1 si comporta invece in modo non deterministico, scegliendo una nuova configurazione C’’ tra un insieme legale e passando il controllo al giocatore 2.

In questo modo si rompe la similitudine di comportamento dei due giocatori: il giocatore 1 si comporta liberamente, mentre il giocatore 2 è vincolato a “leggi di Natura probabilistiche”. Ricorda un po’ il passero solitario del recanate, non libero perché “… di natura è frutto ogni vostra vaghezza …”. Il giocatore 1 cerca di vincere al gioco avendo come avversario ... la Natura.

Un modello generale per questo tipo di gioco è descritto attraverso una *macchina alternante probabilistica polinomiale*:

Definizione 1: una *macchina alternante probabilistica polinomiale*:è una macchina alternante che ha un numero di alternanze polinomiale e in cui il passaggio da una configurazione universale a una esistenziale avviene con un algoritmo randomizzato in tempo polinomiale.

Data una macchina M alternante probabilistica polinomiale ed un ingresso w, il giocatore 1 può decidere una strategia S assegnando ad ogni configurazione esistenziale C una precisa configurazione universale prossima C’. Per ogni strategia S, la macchina M su ingresso w esegue un algoritmo probabilistico ed accetta w con probabilità $p_M(w,S)$: il miglior comportamento del giocatore 1 sta nel scegliere la strategia che massimizza $p_M(w,S)$, con una probabilità di accettazione $p_M(w)$ pari a:

$$p_M(w) = \mathrm{Max}_S\, p_M(w,S)$$

Con una tecnica simile a quella usata per provare che ATIME(poly)$\subseteq$PSPACE, si può provare:

Proposizione 1: per ogni macchina alternante probabilistica polinomiale M, la funzione $p_M(w)$ è calcolabile da un algoritmo deterministico in spazio polinomiale.
Dimostrazione:
Ogni sequenza di calcolo di M su w con |w|=n è di lunghezza al più p(n) per un opportuno polinomio p, quindi ogni configurazione C raggiunta avrà allora lunghezza $|C| \leq p(n)$; ogni configurazione, inoltre, ha al più un numero costante c=O(1) di configurazioni raggiungibili in un passo: tali configurazioni sono ordinabili (ad esempio in ordine lessicografico) in modo tale che la struttura della computazione è, astrattamente, un albero ordinato che ha come nodi le configurazioni, come radice la configurazione iniziale ed è tale che i figli di una configurazione C sono le configurazioni raggiungibili in un passo da C. Alle foglie aggiungiamo come informazione

la loro probabilità di accettazione (1 se la foglia è una configurazione accettante, 0 altrimenti), mentre ai nodi interni C associamo variabili $x_1 \ldots x_c$ tante quante i figli; x_k è destinata a contenere la "probabilità ottima" del sottoalbero individuato dal k° figlio di C .

Questo albero può essere percorso "*in profondità*". Nell'attraversamento dell'albero, il nodo C memorizza quale figlio è correntemente investigato; quando C riceve dal figlio C_k di posto k un numero b (che rappresenta la probabilità ottima del sottoalbero individuato da C_k), se C è esistenziale pone x_k=b, se C è universale pone x_k=b.q dove q è la probabilità di passare da C a C_k . Dopo aver ricevuto le informazioni da tutti i suoi figli, C calcola $\sum_{k=1,c} x_k$ (se C è una configurazione universale) o $Max_{k=1,c}\ x_k$ (se C è una configurazione esistenziale) e invia il risultato al padre.

In questo modo il numero dalla radice dell'albero con il precedente algoritmo è $p_M(w)$. Poiché l'albero ha profondità p(n), l'algoritmo può essere implementato mediante una pila che contiene al massimo p(n) elementi. Poiché sia la lunghezza di ogni configurazione che dei numeri memorizzati è polinomiale in n, il calcolo avviene in spazio polinomiale. □

Le nozioni, che abbiamo proposto con la motivazione di introdurre meccanismi probabilistici nei giochi a informazione completa, sono state in realtà formulate negli anni ottanta, in ambito di complessità strutturale, in relazione allo studio della potenza dell'interazione. Richiamiamo qui brevemente le argomentazioni.

Ricordiamo che un linguaggio L è in NP se è riconoscibile da una Macchina di Turing non deterministica in tempo polinomiale p(n). In particolare:

1. Se $w \in L$ esiste una computazione di M, su ingresso w, accettante e di lunghezza p(n)
2. Se $w \notin L$ qualsiasi computazione di M di lunghezza p(n), su ingresso w, non è accettante

Le computazioni di M su ingresso w formano un albero ordinato binario: ogni computazione può essere quindi descritta da una parola $x \in \{d,s\}^*$ che individua un cammino dalla radice a una foglia di tale albero (d ≈ "figlio destro", s ≈ "figlio sinistro"). Conoscendo x, in tempo p(n) è possibile "controllare" se la computazione denotata da x è accettante o no.

Il linguaggio L può quindi essere riconosciuto attraverso una interazione tra un "Dimostratore" D e un "Verificatore" V con il seguente protocollo:

Ingresso: w

1. Il "dimostratore" D invia al "verificatore" V una parola x di lunghezza polinomiale in |w|
2. Il "verificatore" V, sulla base della conoscenza di w e di x, applica un algoritmo deterministico che, in tempo polinomiale, risponde "accetta" o "rifiuta".

Il criterio di riconoscimento è il seguente:

1. Se $w \in L$ esiste un messaggio x di D tale che V "accetta".
2. Se $w \notin L$ qualsiasi messaggio y di D è tale che V "rifiuta"

Un linguaggio in NP può quindi essere riconosciuto da una *interazione* tra un *verificatore* V, di potenza di calcolo limitata in quanto può lavorare solo *in tempo polinomiale con algoritmi deterministici*, e un *dimostratore* D, che si comporta come un demiurgo, senza limiti alle proprie capacità di calcolo. NP è quindi caratterizzato dal protocollo di interazione e dal criterio di riconoscimento precedentemente descritti.

Che cosa succede se manteniamo il criterio di riconoscimento ma modifichiamo il protocollo, per esempio permettendo un numero di interazioni tra V e D polinomiale in |w|? (cioè D può inviare, anche in modo adattativo, p(n) messaggi x_1 .. $x_{p(n)}$ ognuno dei quali di lunghezza polinomiale, e V può eseguire calcoli e formulare richieste in tempo polinomiale). Ebbene, con lo schema interattivo esteso continuiamo a riconoscere linguaggi in NP: basta infatti che il dimostratore invii all'inizio, una volta per tutte, la parola (x_1 , .. , $x_{p(n)}$) al verificatore, e sulla base di questa parola il verificatore, senza nessun altra interazione, potrà decidere se accettare o meno sempre in tempo polinomiale!.

La situazione è diversa se permettiamo al verificatore di eseguire algoritmi randomizzati. Questo fatto curioso ha posto il problema di studiare la potenza dell'interazione tra un dimostratore e un verificatore probabilistico; un discreto numero di mandarini dell'argomento ha introdotto, nel corso degli anni anni ottanta, variazioni definitorie dei protocolli e dei criteri di riconoscimento, studiandone le relative proprietà.

Ad esempio, il protocollo detto Sistema di Prova Interattivo, introdotto da Goldwasser et al. (S.Goldwasser, S. Micali, and C. Rackoff, "The Knowledge complexity of interactive proof-systems" Proceedings of 17th Symposium on the Theory of Computation, Providence, Rhode Island. 1985), è descritto dallo schema seguente:

Ingresso: una parola w
For k=0,H **do**

- Il verificatore V applica a w, x_1.. x_k un algoritmo randomizzato di al più p(|w|) passi e invia al dimostratore D una richiesta
- Il dimostratore D risponde inviando a V una parola x_k di lunghezza al più p(|w|)

Il verificatore V applica a w, x_1.. x_H un ultimo algoritmo, che termina con "accetta" o "respingi"

Nello schema precedente p(n) è un polinomio e H è il numero di passi (di interazione). Non è difficile riconoscere nel modello una macchina alternante probabilistica M con H alternanze, in cui D è il giocatore 1 e V il giocatore 2. Il criterio di riconoscimento di un linguaggio L è il seguente:

1. Se $w \in L$ allora $p_M(w) = 1$ (cioè: esiste una strategia S di invio di messaggi da parte di D per cui l'algoritmo accetta con certezza)
2. Se $w \notin L$ allora $p_M(w) \leq 1/2$ (cio: per qualsiasi strategia adottata da D, l'algoritmo rifiuta con probabilità almeno 1/2)

Esempio 1:
Il problema di ISOMORFISMO tra grafi richiede di decidere se due grafi F,G sono isomorfi, cioè se esiste una permutazione π dei vertici che trasforma G in F (scriveremo: $F=\pi(G)$. Chiaramente questo problema è in NP: se F e G sono isomorfi, il dimostratore può inviare la permutazione π (di taglia polinomiale) e il verificatore può controllare (in tempo polinomiale) che $F=\pi(G)$. Che cosa si può dire sul problema di NON ISOMORFISMO, che richiede di decidere se due grafi F,G non sono isomorfi? Non è tuttora noto se questo problema sia in NP, e tuttavia può essere risolto dal seguente Sistema di Prove Interattivo di un solo passo:

Ingresso: grafi F,G

- V sceglie una permutazione dei vertici π a caso, sceglie $X \in \{F,G\}$ con probabilità ½ , invia a D il grafo $Y= \pi(X)$
- D invia a V un grafo $Z \in \{F,G\}$

Se X= Z allora V accetta, altrimenti respinge.

Verifichiamo che questo protocollo risolve NON ISOMORFISMO nel senso del criterio di riconoscimento che abbiamo stabilito.

1. G non isomorfo ad H. Se D invia l'unico grafo Z in {F,G} isomorfo a Y allora V "accetta" con certezza.
2. G isomorfo ad H. Pur conoscendo Y, D non ha modo di individuare il grafo X poichè sia F che G sono isomorfi a Y. La probabilità di rifiuto è allora ½ qualsiasi sia la scelta di D. □

Il precedente esempio mostra un problema in coNP (di cui non è nota l'appartenenza a NP) risolubile con un sistema interattivo a un solo passo.

Si osservi che nel protocollo precedentemente introdotto i bit casuali generati da V per eseguire il sua algoritmo randomizzato sono "privati": essi devono essere noti solo a V e non a D. A tal riguardo, osserviamo che in **Esempio 1**, il Verificatore esegue il frammento di programma:
" ….. lancia una moneta, se il risultato è T poni X= F, se è C poni X= G , invia Y= π(X) …."
E' cruciale che il Dimostratore *non* conosca il risultato del lancio della moneta. Cosa succede se i bit casuali sono pubblici, cioè noti anche a D?

A tal riguardo, Babai e Moran (L.Babai, S.Moran, "Arthur-Merlin Games: a Randomized Proof System and a Hierarchy of Complexity Classes", Jour. Comp. Syst. Sci., 36, 254-276, 1988) introducono i cosiddetti "giochi Artù-Merlino", descritti dal precedente protocollo con la richiesta aggiuntiva che i bit casuali usati dal Verificatore siano noti al dimostratore.

Si è in seguito osservato che questa differenza è meno rilevante di quanto possa sembrare: ogni protocollo con una interazione e bit casuali privati può essere trasformato in un equivalente protocollo con un numero costante di interazioni e bit casuali pubblici.

I modelli introdotti differiscono anche rispetto ai criteri di riconoscimento. Per esempio, il criterio di riconoscimento nei giochi "Artù-Merlino" è il seguente:

1. Se $w \in L$ allora $p_M(w) > 2/3$ (cioè esiste una strategia S di invio di messaggi da parte di D per cui l'algoritmo accetta con un buon margine)
2. Se $w \notin L$ allora $p_M(w) < 1/3$ (cioè l'algoritmo rifiuta con un buon margine qualsiasi strategia di invio di messaggi D adotti)

Infine, il criterio di riconoscimento nei "giochi contro natura" introdotti da Papadimitriou (C.Papadimitriou, "Games Against Nature", Jour. Comp. Syst. Sci., 16, 288-301, 1985) è il seguente:

1. Se $w \in L$ allora $p_M(w) > 1/2$
2. Se $w \notin L$ allora $p_M(w) \leq 1/2$

Le differenze tra le capacità computazionali dei tre modelli presentati (Goldwasser, Babai e Papadimitriou) svaniscono quando si considerino protocolli che prevedono un numero di passi polinomiale. A tal riguardo, la classe di problemi risolubili da questi modelli è:

Definizione 2: IP è la classe di problemi risolubili con sistemi di prova interattivi con un numero polinomiale di interazioni

Lo studio della potenza dell'interazione richiede di confrontare IP con le classi di complessità che abbiamo introdotto. Un primo risultato, immediata conseguenza di **Proposizione 1** è la seguente:

Proposizione 2: IP $\subseteq$ PSPACE

12. IP = PSPACE

Abbiamo visto che un Sistema di Prova Interattivo può essere interpretato come un gioco in modo naturale: il giocatore 1 (il Dimostratore) e il giocatore 2 (il Verificatore) interagiscono tra loro modificando lo stato del gioco (la configurazione di calcolo). Tuttavia, mentre il giocatore 1 sceglie la sua possibile mossa in modo non deterministico, nel modello interattivo il giocatore 2 è vincolato a un comportamento probabilistico. Si perde quindi la simmetria di comportamento tra i due giocatori: anziché giocare contro un avversario a lui simile, il giocatore 1 gioca "contro la Natura". Per quanto riguarda il criterio di riconoscimento nel modello interattivo, la parola w viene riconosciuta quando il giocatore 1 ha una strategia vincente, mentre w non è riconosciuta quando il giocatore 1 ha un alta probabilità di perdere qualsiasi strategia persegua.

E' pertanto naturale cercare di confrontare questi due modelli, così differenti "a priori". Il confronto può essere fatto studiando la relazione di inclusioni tra le classi di complessità riferite ai modelli. ATIME (=PSPACE) formalizza i giochi risolubili su macchine alternanti con partite polinomiali in tempo, così come IP è la classe di linguaggi riconoscibili con un numero polinomiale di interazioni. Abbiamo visto nella scora sezione che IP$\subseteq$ PSPACE. In questa sezione proviamo che le due classi coincidono.

Per prima cosa proviamo che il problema di valutare un polinomio espresso in forma compressa mediante speciali programmi lineari è in IP. A tal riguardo, consideriamo polinomi P(x) in indeterminate $x = x_1.. x_n$ e coefficienti in Z_p. Tali polinomi possono essere rappresentati in modo compresso da programmi lineari (PL): un programma lineare di lunghezza N è descritto:

1. da un polinomio P_0
2. da una sequenza di istruzioni $P_k(x) = P_{k-1}(f_k(x)) \sigma_k P_{k-1}(f_k(x))$ (k=1,..,N), con $\sigma_k \in \{+, .\}$ e $f_k(x)$, $g_k(x)$ funzioni calcolabili in tempo polinomiale.

Per semplicità descriveremo un PL con la sequenza $P_N; P_{N-1}; \ldots ; P_0$, sottintendendo gli altri elementi. Il grado d del programma lineare è il massimo dei gradi di $P_k(x)$ (k=1,.,N).
Consideriamo il seguente problema:

Problema: VALUTAZIONE PL
Istanza: un PL $P_N; P_{N-1}; \ldots ; P_0$ di grado d, un elemento $a \in Z_p$, un vettore $z \in Z_p^n$.
Questione: è $a = P_N(z)$?

Proposizione 1: Se $p \geq 2Nd$, il seguente Sistema Interattivo risolve VALUTAZIONE PL:

Procedura $F(P_k; P_{k-1}; \ldots; P_0, z, a)$
If k=0 **then**

- V esegue**: if** $P_0(z)=a$ **then** "accetta" **else** "rifiuta"

If k>0 **then**

- D invia a V un polinomio $h_k(t)$ in una variabile t di grado al più d
- V controlla se $a=h(0) \sigma_k h(1)$
  If NO **then** "rifiuta"
  If SI **then**
  a. V estrae a caso $t \in Z_p$
  b. $z' = t_k f_k(z) + (1-t).g_k(z)$; $a'=h(t_k)$
  c. Procedura $F(P_{k-1}; P_{k-2}; \ldots; P_0, z', a')$

Dimostrazione:
Per provare che il Sistema Interattivo risolve correttamente il problema VALUTAZIONE PL, analizziamo separatamente i due casi $P_k(z)=a$ e $P_k(z)\neq a$:

$P_k(z)=a$.

Per ogni k>0, basta che D invii a V il polinomio $h(t) = P_{k-1}(tf_k(z) + (1-t).\ g_k(z))$. Tale polinomio è di grado al più d, inoltre vale che:

1. $h(0)\ \sigma_k\ h(1) = P_{k-1}(g_k(z))\ \sigma_k\ P_{k-1}(f_k(z)) = a$
2. $a' = h(t_k) = P_{k-1}(t_k f_k(z) + (1-t_k).\ g_k(z)) = P_{k-1}(z')$

Viene quindi richiamata la procedura F su P_{k-1}; P_{k-2}; …; P_0, z', a', in cui $P_{k-1}(z')=a'$. Induttivamente, ci si riduce al caso k=0 in cui la procedura accetta. Esiste quindi una strategia di D che porta ad accettare con certezza.

$P_k(z)\neq a$.

Indichiamo con Φ_k la probabilità che, fissata una qualsiasi strategia di D, Procedura F(P_k; P_{k-1}; …; P_0, z, a) termini con rifiuto sapendo che $P_k(z)\neq a$.
Se V riceve h(t) tale che $a \neq h(0)\ \sigma_k\ h(1)$, allora la computazione termina immediatamente con rifiuto.Se V riceve h(t) tale che $a = h(0)\ \sigma_k\ h(1)$, allora deve essere $h(t) \neq P_{k-1}(tf_k(z) + (1-t).g_k(z))$; se così non fosse, sarebbe $a = h(0)\ \sigma_k\ h(1) = P_{k-1}(g_k(z))\ \sigma_k\ P_{k-1}(f_k(z)) = P_k(z)\neq a$, assurdo. Ne segue che il polinomio $h(t) - [P_{k-1}(tf_k(z) + (1-t).\ g_k(z))]$ ha al più d radici e quindi, se p sufficientemente grande, la probabilità che un elemento di Z_p sia una radice è al più d/p. Ne segue che, con probabilità almeno 1-d/p, un elemento t_k estratto a caso è tale che $h(t_k) \neq P_{k-1}(t_k f_k(z) + (1-t_k).\ g_k(z))$.
Viene quindi richiamata la procedura F su P_{k-1}; P_{k-2}; …; P_0, z', a', in cui $P_{k-1}(z') \neq a'$, quindi:
(*) $\Phi_k \geq (1-d/p)\Phi_{k-1}$
Si conclude che $\Phi_k \geq (1-d/p)^N \geq 1-Nd/p$. Se $p\geq 2Nd$, qualsiasi strategia di D porta a rifiutare con probabilità almeno ½. □

Il problema VALUTAZIONE PL è un problema particolarmente significativo per PSPACE. Infatti:

Proposizione 2: VALUTAZIONE PL è PSPACE-difficile.
Dimostrazione:
Dato un linguaggio L in PSPACE, esiste una macchina di Turing deterministica M che riconosce L in spazio polinomiale p(n). Sia w una parola di ingresso, con |w|=n. Indicando con $\sum$ è l'insieme di lettere e stati di M, codifichiamo gli elementi di $\sum$ con 0, 1, …, $|\sum|$ visti come elementi di Z_p. Una configurazione raggiungibile da quella iniziale q_0w sarà del tipo: $C= x_1x_2 \dots x_{p(n)}$, con $x_k \in Z_p$.

Consideriamo ora un polinomi multivariati $P_k(x,y)$, con $x= x_1x_2 \dots x_{p(n)}$ e $y= y_1y_2 \dots y_{p(n)}$ tali che, se sia $x_1x_2 \dots x_{p(n)}$ che $y_1y_2 \dots y_{p(n)}$ sono configurazioni di M, valga:

$P_k(x,y)$ = **se** y raggiungibile da x in 2^k passi di calcolo **allora** 1 **altrimenti** 0

Poiché M lavora al più in tempo $|\sum|^{p(n)}$, posto $N = |\sum| \log p(n)$ e dette in(w), acc rispettivamente la codifica della configurazione iniziale e di quella accettante, vale:

$w \in L$ se e solo se $P_N(in(w),acc)=1$

Se riusciamo a costruire in tempo polinomiale un programma lineare per $P_N(x,y)$, abbiamo ridotto L a VALUTAZIONE PL, mostrando che quest'ultimo problema è PSPACE-difficile.

A questo riguardo ricordiamo che $y= y_1y_2 \dots y_{p(n)}$ è raggiungibile in un passo da $x= x_1x_2 \dots x_{p(n)}$ se e solo se $y_k = g(x_{k-2}x_{k-1}x_k\ x_{k+1}x_{k+2})$ per una opportuna funzione parziale g e per $1\leq k\leq p(n)$.

Con tecniche standard di interpolazione, possiamo costruire un polinomio $\Phi(x_1x_2x_3x_4x_5)$ in 5 variabili e di grado costante c tale che:

$$\Phi(yx_1x_2x_3x_4x_5) = \textbf{se } y= g(x_1x_2x_3x_4x_5) \textbf{ allora } 1 \textbf{ altrimenti } 0$$

Siamo quindi in grado di ottenere $P_0(x,y)$ come il seguente polinomio di grado O(p(n)):

$$(*) \qquad P_0(x,y) = \prod k=1,p(n)\ \Phi(y_k x_{k-2} x_{k-1} x_k x_{k+1} x_{k+2})$$

Inoltre, vale per ogni k:

$$(**) \qquad P_k(x,y) = \sum_z P_{k-1}(x,z).\ P_{k-1}(z,y)$$

Non possiamo interpretare (**) come un programma lineare perchè il calcolo di P_k a partire da P_{k-1} richiede un numero esponenziale di somme (le configurazioni z nella somma sono generalmente in numero esponenziale). Allo scopo di ridurre tali somme, osserviamo intanto che codificando in binario i simboli di $\sum$ la configurazione z è codificata da una parola binaria *z* di lunghezza $N = p(n) \log |\sum|$.

Posto: $Q_{kN}(x,y, z_1 z_2 \ldots z_N) = P_{k-1}(x,z).\ P_{k-1}(z,y)$
da (**) segue $P_k(x,y) = \sum_{z1\in\{0,1\}} \sum_{z2\in\{0,1\}} \sum \ldots \sum_{zN\in\{0,1\}} Q_{kN}(x,y, z_1 z_2 \ldots z_N)$

Posto infine : $Q_{kj}(x,y, z_1 z_2 \ldots z_j) = Q_{kj+1}(x,y, z_1 z_2 \ldots z_j,0) + Q_{kj+1}(x,y, z_1 z_2 \ldots z_j,1) \quad (j=0,N-1)$
Risulta : $Q_{k0}(x,y) = \sum_{z1\in\{0,1\}} \ldots \sum_{zN\in\{0,1\}} Q_{kN}(x,y, z_1 z_2 \ldots z_N) = P_k(x,y)$

$P_N(x,y)$ è allora calcolabile con il programma lineare dato dal polinomio $P_0(x,y)$ e dalla sequenza di istruzioni (delle quali scriviamo per semplicità solo la parte sinistra):

$$P_N ; Q_{N1} ; .. Q_{NN} ; P_{N-1}; Q_{N-11}; ..; Q_{N-1N} ;\ldots; P_1 ; Q_{11} ; .. Q_{1N}$$

Ogni polinomio $P_k(x,y)$ è di grado O(p(n)) e ogni polinomio $Q_{kj}(x,y, z_1 z_2 \ldots z_j)$ è al più di grado doppio, quindi ancora O(p(n)). Il programma lineare ha infine lunghezza N^2, quindi $O(p(n)^2)$.
Si conclude che VALUTAZIONE PL è un problema PSPACE-difficile. □

Possiamo concludere col principale risultato di questa sezione, dovuto a Shamir, che dà un ulteriore conferma della robustezza della classe PSPACE:

Proposizione 3: IP=PSPACE
Dimostrazione:
Il problema VALUTAZIONE PL è PSPACE-difficile (Proposizione 2) e contemporaneamente è in IP (proposizione 1); questo implica che PSPACE $\subseteq$ IP. Poiché in precedenza avevamo mostrato che PSPACE $\supseteq$ IP, segue la tesi.

E.R. Berlekamp, J.H. Conway, R.K. Guy, "Winning ways for your mathematical plays" , **I–II** , Acad. Press (1982)