# Relazione Percorso d'Eccellenza

Leonardo Danella

Anno Accademico 2020/2021

## 1 Introduzione

In questo Percorso d'Eccellenza, svolto sotto la guida del Professor *Nicola Galesi* nell'anno accademico 2020/2021, è stato trattato il ben noto problema conosciuto come "$P = NP$?", formulato negli anni '70 da Stephen Cook dell'Università di Toronto. Informalmente tale problema chiede di dimostrare o refutare se computazioni deterministiche efficienti siano sufficienti per catturare computazioni non-deterministiche efficienti. In tale contesto con computazioni efficienti si intendono algoritmi il cui tempo di esecuzione sia limitato polinomialmente. Ad esempio, supponiamo che si stiano organizzando gli alloggi per un gruppo di quattrocento studenti universitari. Lo spazio è limitato e solo cento degli studenti riceveranno posti nel dormitorio. Per complicare le cose, il rettore ha fornito una lista di coppie di studenti incompatibili e ha richiesto che nessuna coppia di quest'ultima appaia nella scelta finale. Questo è un esempio di ciò che viene chiamato un problema $NP$, poiché è facile controllare se una data scelta di cento studenti proposta da un collega sia soddisfacente (cioè che nessuna coppia presa dalla lista del tuo collega appaia anche nella lista dell'ufficio del Rettore), tuttavia il compito di generare una tale lista da zero sembra essere così difficile da essere completamente impraticabile. Infatti, il numero totale di modi di scegliere cento studenti tra i quattrocento candidati $\binom{400}{100}$ è maggiore del numero di atomi nell'universo conosciuto! Quindi nessuna civiltà futura potrebbe mai sperare di costruire un supercomputer capace di risolvere il problema utilizzando la forza bruta, cioè controllando ogni possibile combinazione. Tuttavia, questa apparente difficoltà potrebbe solo riflettere la mancanza di ingegno del programmatore. Infatti, uno dei problemi più importanti dell'informatica è determinare se esistono domande la cui risposta può essere controllata rapidamente, ma che richiedono un tempo impossibilmente lungo per essere risolte con qualsiasi procedura diretta. Problemi come quello elencato sopra sembrano certamente essere di questo tipo, ma finora nessuno è riuscito a dimostrare che qualcuno di essi sia davvero così difficile come sembra, cioè che non esista davvero un modo fattibile per generare una risposta con l'aiuto di un computer. Stephen Cook e Leonid Levin hanno formulato il problema $P$ (cioè facile da trovare) contro $NP$ (cioè facile da controllare) indipendentemente nel 1971. Più formalmente possiamo definire le due classi $P$ ed $NP$ come segue:

$P$ è l'insieme dei linguaggi che sono decidibili in tempo polinomiale su una macchina di Turing deterministica a nastro singolo. Cioè:

$$P = \bigcup_{k \in \mathbb{N}} TIME(n^k)$$

$NP$ è l'insieme dei linguaggi che sono decidibili in tempo polinomiale su una macchina di Turing non deterministica. Cioè:

$$NP = \bigcup_{k \in \mathbb{N}} NTIME(n^k)$$

Inoltre abbiamo introdotto la classe di complessità $coNP$. Un problema decisionale X è un membro di $coNP$ se e solo se il suo complemento $\bar{X}$ è nella classe di complessità NP. In maniera più formale si ha che se $S$ è un problema su un alfabeto $A$ allora esso è un problema della classe $coNP$ se e solo se $A^* \backslash S = S^c$ è un problema di classe $NP$.

# 2 Il problema SAT, TAUT, Sistema di Dimostrazioni

## 2.1 SAT

In informatica e logica, il problema di soddisfacibilità booleana, anche chiamato soddisfacibilità proposizionale o SAT, è il problema di determinare se esista o meno un assegnamento che soddisfi una data formula booleana. Ad esempio la formula $A \wedge \neg B$ è soddisfacibile poiché possiamo trovare i valori A = True e B = False che rendono $A \wedge \neg B = TRUE$. Invece $A \wedge \neg A$ è insoddisfacibile (UNSAT).

## 2.2 TAUT

Una tautologia, in logica, è una formula che viene interpretata come True in ogni sua possibile interpretazione.

## 2.3 Sistema di dimostrazioni

### 2.3.1 Definizione PPS

Un sistema di dimostrazione proposizionale (PPS) $S$ per un certo linguaggio $L$ (TAUT) si può definire con una funzione suriettiva:
$f_S : \{0,1\}^* \implies$ TAUT calcolabile in P-time, ovvero in tempo polinomiale.
I PPS si dividono in due sistemi principali: Deduttivo e Refutazionale.

### 2.3.2 Sistema Deduttivo

Un sistema di dimostrazione deduttivo permette di dimostrare una tautologia non banale partendo da istanze di assiomi triviali usando regole perlopiù standardizzate, quali ad esempio il modus ponens.

### 2.3.3 Sistema Refutazionale (esempio risoluzione)

In un sistema di dimostrazione refutazionale si parte dalla negazione di una tautologia che si vuole dimostrare e si vede se si raggiunge, applicando la Regola di Risoluzione, la clausola vuota.
La regola di risoluzione è definita come segue:

$$(C \vee x) \wedge (\bar{x} \vee D) \implies C \vee D$$

Una refutazione in Risoluzione della formula $F$ in forma $CNF$ è una sequenza di clausole $C_1, C_2, ..., C_m$tale che $\forall i = 1, ..., m$:

1. $C_i$ è una clausola di F, oppure
2. $C_i$ è ottenuta da $C_j$ e $C_k$ dove $j, k < i$ tramite la regola di Risoluzione,
3. $C_m = \square$.

Una refutazione $C_1, C_2, ..., C_m$ può essere vista come un grafo diretto e aciclico (dag)

$$G \equiv (V, E).$$

dove $V = \{C_1, C_2, ..., C_n\}$ nodi sorgente saranno relativi alle clausole della formula mentre il pozzo sarà la clausola vuota derivata dalla dimostrazione ($\square$), e un arco diretto collega le premesse di una regola con la conclusione.
Di un sistema di dimostrazione è necessario poter dimostrare correttezza e completezza. Nel caso del sistema refutazionale, ad esempio, si dimostrano nel seguente modo:

1. Correttezza (soundness)
   secondo la quale se la formula $X$ è una tautologia, allora $X$ ha una prova di risoluzione la cui conclusione è una conseguenza logica delle premesse, viene dimostrata partendo dal fatto che la regola di risoluzione è corretta (sound), cioè se

   $$(\alpha \models C \vee x) \wedge (\alpha \models D \vee \bar{x}) \implies (\alpha \models C \vee D)$$

   e procedendo per assurdo.

2. Completezza (completeness)
   secondo la quale se la formula $X$ è una tautologia, allora possiamo essere sicuri che dopo un numero finito di applicazioni della regola di risoluzione partendo da $\neg X$ raggiungiamo la clausola vuota $\square$, si dimostra per induzione su $n$, che rappresenta il numero di variabili nella formula X.

# 3 Sistema di dimostrazioni "poly-bounded"

Un sistema di dimostrazioni proposizionale (PPS) è una relazione uno a uno da confutazioni a formule insoddisfacibili controllabili in tempo polinomiale.

Un $PPS$ P è definito *polinomialmente delimitato* se per ogni $k - CNF$ $\tau$ insoddisfacibile con $n$ variabili e poly(n) clausole, esiste una prova $P\pi$ tale che $|\pi| \leq poly(n)$.

# 4 Th. Cook-Reckhow

$NP = coNP \iff \exists$ un PPS poly-bounded
Dimostrazione[1]:
NP è esattamente l'insieme dei linguaggi con sistemi di dimostrazione p-bounded

$(\implies)$
$TAUT \in coNP$ poiché: $F$ non è una tautologia $\iff \neg F \in SAT$.
Se $NP = coNP$, allora $TAUT \in NP$ ed ha un sistema di dimostrazioni p-bounded per definizione.
$(\impliedby)$
Supponiamo che esista un sistema di dimostrazione p-bounded.
Allora $TAUT \in NP$, e poiché $TAUT$ è completo per $coNP$ segue che $NP = coNP$.

# 5 Tree-like RES, Tree-like RESxor(+)

Una refutazione in Risoluzione è ad albero se ogni clausola nella dimostrazione viene utilizzata al più una volta all'interno della regola di risoluzione; cioè una refutazione è in albero se il grafo a essa associata è un albero.
Per poter capire la potenza della risoluzione in albero è necessario introdurre il prover-delayer game di Pudlàk ed Impagliazzo [2], uno dei metodi canonici per le dimostrazioni dei lower-bound.
Il Prover-Delayer game di Pudlák e Impagliazzo nasce dal noto fatto che una prova di risoluzione ad albero per una formula F può essere vista come un albero decisionale che risolve il problema di ricerca di trovare una clausola di F falsificata da un assegnazione data. Nel gioco, il Prover interroga una variabile e il Delayer o gli dà un valore o lascia la decisione al Prover e riceve un punto. Il numero di punti del Delayer alla fine del gioco è quindi proporzionale all'altezza dell'albero delle prove. È facile argomentare che mostrare limiti inferiori di dimostrazioni con questo gioco funziona solo se il grafo di ogni refutazione di risoluzione ad albero contiene un sottoalbero bilanciato minimo, e l'altezza di quel sottoalbero fornisce il limite inferiore della dimensione.[3]

## 5.1 Th. 1

Una domanda che sorge spontanea è quanti punti debba lasciare il Prover al Delayer per vincere nel *caso peggiore*.
Per rispondere a questa domanda è stato formulato il teorema secondo il quale:

#### 5.1.1 *Se F, formula CNF, UNSAT ha una refutazione in TreeRES di dimensione S (ovvero con S clausole), allora il prover vince il Prover-Delayer game su F lasciando al più O(log(S)) punti al Delayer.*

Questo teorema viene dimostrato con l'uso dell'invariante ed è di grande importanza poiché viene utilizzato per trovare un limite inferiore per il Pigeon Hole Principle (PHP).

### 5.2 Th. 2

Questo teorema afferma che:

#### 5.2.1 $\exists$ *una famiglia di strategie per il Delayer tale che contro ogni strategia del Prover sul* $PHP_n^{n+1}$ *il Delayer guadagna* $\geq \Omega(n)$ *punti*

Per dimostrare un limite inferiore del $PHP_n^{n+1}$ sfruttiamo quest'ultimo teorema affermando che:

### 5.3 Corollario Th. 2

Il $PHP_n^{n+1}$ richiede in Tree-RES dimostrazioni di dimensione $2^{\Omega(n)}$

#### 5.3.1 Dimostrazione 5.3

Questo corollario si dimostra semplicemente procedendo per assurdo, infatti assumendo che $\exists$ una dimostrazione $\Pi$ in Tree-Res di dimensione $S$ tale che $S < 2^n$ per il $PHP$, allora avremo per il teorema **5.1** il Prover vince lasciando $< O(log(2^n))$ \$ al Delayer.

## 6 PHP

Il Principio della Piccionaia ($PHP$) afferma che se $n$ oggetti sono messi in $m$ contenitori, con $n > m$, allora almeno un contenitore deve contenere più di un oggetto. Questa affermazione apparentemente ovvia, può essere usata per dimostrare risultati inaspettati. Per esempio, dato che la popolazione di Londra è maggiore del numero massimo di capelli che possono essere presenti sulla testa di un uomo, allora per il $PHP$ sappiamo che ci sono almeno due persone a Londra che hanno lo stesso numero di capelli sulla testa.
Definiamo poi due assiomi del $PHP$ che devono valere in ogni momento:

- l'assioma del piccione, secondo il quale ogni piccione deve essere in una gabbia:

$$\bigvee_{j \in [n]} x_{i,j} \qquad \forall i \in [m]$$

- l'assioma della gabbia, secondo il quale ogni gabbia contiene al più un piccione:

$$\neg x_{i',k} \vee \neg x_{i'',k} \quad con \quad k \in [n] \quad e \quad i', i'' \in [m] \; con \quad i' \neq i''$$

# 7 il PHP in Tree-like RES($\oplus$)

## 7.1 Alberi di parità

Concludiamo la relazione trattando gli alberi di parità (Parity Decision Trees). Con albero di parità intendiamo un albero binario $T$ i cui archi sono uguaglianze lineari.[4]
Allora per ogni $v$ vertice appartenente a $T$, definiamo $\Phi_v^T$ il sistema delle uguaglianze del percorso dalla radice a $v$. Quindi il Parity decision tree su una formula $CNF$ $\varphi$ è un albero binario nel quale:

- ogni nodo interno è segnato con una forma lineare che dipende da variabili di $\varphi$

$$\wedge$$

- per ogni nodo interno segnato con una forma lineare $f$, uno degli archi verso il figlio è segnato con $f = 0$ e l'altro con $f = 1$.

Per ogni foglia $v$ avremo esattamente una delle seguenti condizioni:

1. $\Phi_v^T$ non ha soluzione. La chiameremo foglia *degenerata*;
2. $\Phi_v^T$ è $SAT$ ed esiste una clausola $C$ di $\varphi$ tale che ogni soluzione di $\Phi_v^T$ falsifica $C$. Diremo che questa foglia *refuta* $C$;
3. $\Phi_v^T$ ha un'unica soluzione tra le variabili di $\varphi$, e questa soluzione soddisfa $\varphi$. Chiameremo questa foglia *satisfying* (che soddisfa $C$).

## 7.2 Res($\oplus$)

Una dimostrazione in Res($\oplus$) è Tree-like se le clausole lineari possono essere organizzate come un albero dove:

- la radice è la clausola vuota ($\square$),
- ogni foglia è una clausola della formula iniziale,
- la clausola lineare ad ogno nodo interno può essere ottenuto dai suoi figli attraverso l'applicazione di una regola.

## 7.3 Limite inferiore per $PHP_n^m$ in Tree-Res($\oplus$)

Dimostriamo ora il risultato di Itsykson e Sokolov [5] secondo il quale il *PHP* richiede dimostrazioni esponenziali anche in tale sistema di dimostrazione. Prendiamo in considerazione ancora una volta il *Principio della piccionaia* e cerchiamo un limite inferiore sulla dimensione dell'albero di decisione in TreeRes($\oplus$) per una formula normale congiuntiva insoddisfacibile $\phi$.
Sfruttiamo ancora una volta il Prover-Delayer game, dando ai due giocatori la formula $\phi$. Questo problema si differenzia dal precedente poiché, nonostante si parta sempre dal sistema lineare vuoto, il Delayer dovrà decidere se assegnare un valore $\alpha : \{0,1\}^*$ all'equazione lineare $f = \alpha$, o se far scegliere l'assegnamento al Prover, guadagnando un punto. Anche questa volta, l'obiettivo del Delayer è quello di massimizzare il numero di punti guadagnati, mentre quello del Delayer è di minimizzarli.
Andiamo ora ad esplicitare dei lemmi necessari per dimostrare il limite inferiore:

### 7.3.1 Se per una formula $CNF$ $\varphi$ insoddisfacibile esiste una strategia per il Delayer che gli permette di guadagnare almeno $t$ punti, allora la dimensione di qualunque Tree-like Res($\oplus$) per $\varphi$ è almeno $2^t$.

Per dimostrarlo consideriamo un albero di parità $T$ per la formula $\varphi$. In questa simulazione avremo che il Delayer procede con una strategia probabilistica, mentre il Prover seguirà la seguente strategia randomica: il Prover parte dalla radice dell'albero $T$ e ad ogni turno dà una formula lineare $f$ al Delayer. Quest'ultimo, utilizzando la sua strategia o decide un valore per $\alpha \in \{0,1\}$, o lascia la scelta al Prover, che deciderà casualmente, guadagnando un punto. Il gioco termina quando si raggiunge una foglia. A questo punto sappiamo per ipotesi che la strategia del Delayer gli ha permesso di guadagnare almeno $t$ punti. Perciò la probabilità che il gioco termini su una foglia è al più $2^{-t}$. Sapendo che, per come è strutturato il gioco, la probabilità che una foglia sia il punto di arrivo è 1, il numero di foglie di $T$ è $2^t$.□
Definiamo un assegnamento a variabili $p_{i,j}$ *proprio* se soddisfa tutti gli assiomi delle gabbie, cioè se in ogni assegnamento proprio non ci sono gabbie con due o più piccioni.

### 7.3.2 Sia $A_p = b$ un sistema lineare con variabili $p = (p_{i,j})_{i\in[m],j\in[n]}$ e con al massimo $\frac{n-1}{2}$ equazioni e abbia questo una soluzione propria. Allora per ogni $i \in [m]$ il sistema ha una soluzione propria che soddisfa l'assioma del piccione $p_{i,1} \vee p_{i,2} \vee ... \vee p_{i,n}$

Per dimostrare partiamo dalla considerazione che la modifica di "uni" in "zeri" in una soluzione propria mantiene propria la soluzione.Ora supponiamo che il sistema lineare abbia $k \leq \frac{n-1}{2}$ equazioni. Consideriamo ora una soluzione $\pi$ del sistema $Ap = b$ che abbia il minimo numero di uni. Vogliamo dimostrare che questo numero minimo sia al più $k$.

Procediamo per assurdo supponendo che si abbiano $k+1$ variabili

$$p_{r,1}, p_{r,2}, ..., p_{r,k+1}$$

che assumano il valore 1 in $\pi$. Sapendo che la nostra matrice ha $k$ righe, avremo che le colonne corrispondenti a $p_{r,1}, p_{r,2}, ..., p_{r,k+1}$ sono linearmente dipendenti. Allora la soluzione $\pi'$ del sistema omogeneo associato $Ap = 0$ esiste, e le variabili poste ad 1 fanno sempre parte dell'insieme $p_{r,1}, p_{r,2}, ..., p_{r,k+1}$. Ora, sapendo che anche $\pi + \pi'$ è soluzione di $Ap = b$, e che è propria poiché sommando le soluzioni avremo un insieme di soluzioni ottenibile prendendo $\pi$ e cambiando degli "uni" in "zeri". In questo modo contraddiciamo la minimalità di $\pi$.
A questo punto sapendo che $\pi$ ha al più $k$ uni possiamo evincere che avrà almeno $n-k$ gabbie vuote. Dato che per ipotesi $k \leq \frac{n-1}{2}$, allora possiamo affermare facilmente che $n - k \geq k + 1$. Scegliamo esattamente $k+1$ gabbie vuote che chiamiamo $l_1, l_2, ..., l_{k+1}$ e fissiamo un $i \in [m]$. Le colonne di $A$ corrispondenti alle variabili

$$p_{i,l_1}, p_{i,l2}, ..., p_{i,lk+1}$$

sono linearmente dipendenti, quindi per lo stesso ragionamento della dimostrazione per assurdo sappiamo che esiste una soluzione $\tau$ del sistema omogeneo associato $Ap = 0$ tale che ogni variabile posta ad 1 in $\tau$ appartenga all'insieme $p_{i,l_1}, p_{i,l2}, ..., p_{i,lk+1}$. L'assegnamento $\pi + \tau$ è una soluzione del sistema $Ap = b$ ed è propria, poiché le gabbie numerate $l_1, l_2, ..., l_{k+1}$ sono state scelte in modo da essere vuote in $\pi$, perciò $\tau$ mette al più un piccione per gabbia. Per concludere va detto che l'assegnamento $\pi + \tau$ soddisfa l'assioma del piccione, poiché non avremo mai due piccioni in una stessa gabbia per costruzione.

### 7.3.3 Per ogni $m > n$, ogni albero di decisione($\oplus$) per il $PHP_n^m$ ha una grandezza di almeno $2^{\lceil \frac{n}{2} \rceil}$

Diciamo che un sistema lineare $\Phi$ implica propriamente un'uguaglianza $f = \alpha$ se ogni soluzione proprio di $\Phi$ soddisfa $f = \alpha$. Per dimostrare questo teorema mostreremo una strategia del Delayer che gli permette di guadagnare almeno $\lceil \frac{n}{2} \rceil$ punti.
Partiamo come sempre da $\Phi$ sistema vuoto, riempendolo di tutte le equazioni $f = \alpha$ con gli assegnamenti decisi dal Prover o dal Delayer nel corso della partita. La strategia è la seguente: se si lascia il Prover decidere una forma lineare $f$, allora se $\Phi$ implica propriamente $f = \alpha$ per un qualche $\alpha \in \{0, 1\}$, allora il Delayer decide di assegnare proprio $\alpha$, altrimente lascia la scelta al Prover.
Ora vogliamo dimostrare che il sistema $\Phi$ ha sempre una soluzione propria (consideriamo questa definizione la nostra invariante). Lo dimostriamo per induzione sul numero dei passi (su ogni turno giocato). L'invariante è sempre vera per il sistema vuoto . Assumiamo che il Prover scelga una formula lineare $f$. Se $\Phi$ ha una soluzione propria, allora o $\Phi \wedge (f = 0)$ o $\Phi \wedge (f = 1)$ ha una soluzione propria. Se così non fosse, ci troveremmo in un caso in cui $\Phi \wedge (f = \alpha)$ non ha soluzione propria, con $\alpha \in \{0, 1\}$. Ma allora avremmo che $\Phi$ implica propriamente $f = 1 + \alpha$. Ci troviamo quindi nel caso descritto prima in cui è il Delayer

a scegliere, e deciderà proprio il valore $1+\alpha$, in modo tale che $\Phi \wedge (f = 1+\alpha)$ abbia una soluzione propria.
Consideriamo le seguenti tre situazioni alla fine del gioco:

- Il sistema $\Phi$ diventa insoddisfacibile. Questo caso è impossibile, dal momento che $\Phi$ ha una soluzione propria
- Il sistema $\Phi$ contraddice l'assioma della gabbia. Anche questa situazione è impossibile, sempre poiché $\Phi$ ha una soluzione propria.
- Il sistema $\Phi$ contraddice l'assioma del piccione $(p_{i,1} \vee p_{i,2} \vee ... \vee p_{i,n})$. Consideriamo $\Phi'$ un sottosistema di $\Phi$ che corrisponde al momento in cui il Delayer lascia la scelta al Prover. Per costruzione ogni equazione da $\Phi$ è implicata propriamente da $\Phi'$. Allora $\Phi'$ non ha alcuna soluzione propria che soddisfi $p_{i,1} \vee p_{i,2} \vee ... \vee p_{i,n}$.
  A questo punto per **7.3.2.** sappiamo che $\Phi'$ consiste in più di $\frac{n-1}{2}$ equazioni. Dato che il Delayer guadagna un punto per ogni equazione di $\Phi'$, sappiamo che guadagnerà con certezza più di $\frac{n-1}{2}$ punti. Si può perciò concludere, per **7.3.1.** che la grandezza di ogni albero di decisione($\oplus$) per il $PHP_n^m$ è di almeno $2^{\lceil \frac{n}{2} \rceil}$.

# References


[1] S. A. Cook and R. A. Reckhow, "The relative efficiency of propositional proof systems," *Journal of Symbolic Logic*, vol. 44, no. 1, p. 36–50, 1979.

[2] P. Pudlák and R. Impagliazzo, "A lower bound for dll algorithms for ¡i¿k¡/i¿-sat (preliminary version)," in *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, (USA), p. 128–136, Society for Industrial and Applied Mathematics, 2000.

[3] O. Beyersdorff, N. Galesi, and M. Lauria, "A lower bound for the pigeonhole principle in tree-like resolution by asymmetric prover–delayer games," *Information Processing Letters*, vol. 110, no. 23, pp. 1074–1077, 2010.

[4] J. Krajíček, *Proof Complexity*. Encyclopedia of Mathematics and its Applications, Cambridge University Press, 2019.

[5] D. Itsykson and D. Sokolov, "Lower bounds for splittings by linear combinations," in *Mathematical Foundations of Computer Science 2014* (E. Csuhaj-Varjú, M. Dietzfelbinger, and Z. Ésik, eds.), (Berlin, Heidelberg), pp. 372–383, Springer Berlin Heidelberg, 2014.