Il problema SAT ed il suo impatto

Agostino Dovier

Dept of Mathematics, Computer Science, and Physics
University of Udine
Udine (Italy)

UDINE, 13 Giugno 2019

Sommario

Vedremo

- il problema SAT
- il suo ruolo nel problema (del millennio) aperto P vs NP
- gli approcci (esponenziali nel caso peggiore) alla sua risoluzione
- come si possano sfruttare positivamente i risultati di tali approcci per risolvere altri problemi
- un linguaggio per modellare esattamente i problemi in NP

Cenni di algebra di Boole

- Avete usato variabili nelle equazioni e nei sistemi
- Per quelle variabili immaginavate possibili valori nei numeri reali $(10, -32, -2.345, \frac{\pi}{2}, \sqrt{2} - \pi, \dots)$
- Una variabile Booleana può assumere invece solo due valori: 0 e 1 (falso e vero, spento e acceso, etc)
- Per questo tipo di variabili e valori ci sono le seguenti operazioni: $\vee$ (or, disgiunzione, che ricorda il $+$), $\wedge$ (and, congiunzione, che ricorda il $\cdot$) e $\neg$ (not, negazione, che ricorda il meno "unario")
- Le operazioni, definite sui valori 0 e 1, sono molto semplici:
  $0 \vee 0 = 0, 0 \vee 1 = 1, 1 \vee 0 = 1, 1 \vee 1 = 1$
  $0 \wedge 0 = 0, 0 \wedge 1 = 0, 1 \wedge 0 = 0, 1 \wedge 1 = 1$
  $\neg 0 = 1, \neg 1 = 0$

SAT

- Dato un insieme di variabili Booleane $\mathcal{X} = \{X_1, \ldots, X_n\}$
- e una formula costruita su $\mathcal{X}$ del tipo:

$$\Phi = (\ell_1^1 \vee \cdots \vee \ell_{n_1}^1) \wedge \cdots \wedge (\ell_1^k \vee \cdots \vee \ell_{n_k}^k)$$

  dove ogni ℓ_j^i è una variable X_p o la sua negazione $\neg X_p$
- Ad esempio $\mathcal{X} = \{X_1, X_2, X_3\}$ e

$$(X_1 \vee \neg X_2) \wedge (\neg X_1 \vee X_2 \vee \neg X_3) \wedge (\neg X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_2 \vee \neg X_3)$$

- Il problema SAT (satisfiability) è quello di stabilire l'esistenza di un assegnamento di valori (0/1) alle variabili in grado di rendere vera la formula Φ

SAT
Esempio

$$(X_1 \vee \neg X_2) \qquad (\neg X_1 \vee X_2 \vee \neg X_3) \qquad (\neg X_1 \vee \neg X_2 \vee X_3) \qquad (\neg X_1 \vee \neg X_2 \vee \neg X_3)$$

1. Data una possibile soluzione, verificarla è facile.
2. Le possibili soluzioni sono "tante" $(2^{|\mathcal{X}|})$.

SAT
ESEMPIO

$(X_1 \vee \neg X_2)$ $(\neg X_1 \vee X_2 \vee \neg X_3)$ $(\neg X_1 \vee \neg X_2 \vee X_3)$ $(\neg X_1 \vee \neg X_2 \vee \neg X_3)$

$X_1 = 1$

1. Data una possibile soluzione, verificarla è facile.
2. Le possibili soluzioni sono "tante" $(2^{|\mathcal{X}|})$.

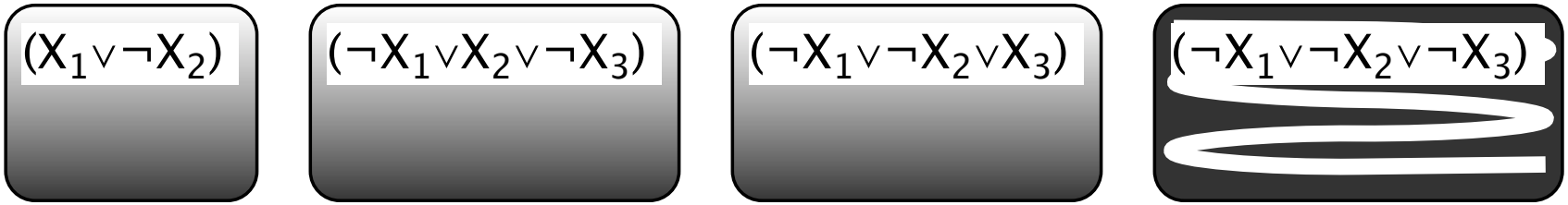
SAT
ESEMPIO

$(X_1 \vee \neg X_2)$ $(\neg X_1 \vee X_2 \vee \neg X_3)$ $(\neg X_1 \vee \neg X_2 \vee X_3)$ $(\neg X_1 \vee \neg X_2 \vee \neg X_3)$

$X_1=1$ $X_2=1$

1. Data una possibile soluzione, verificarla è facile.
2. Le possibili soluzioni sono "tante" $(2^{|\mathcal{X}|})$.

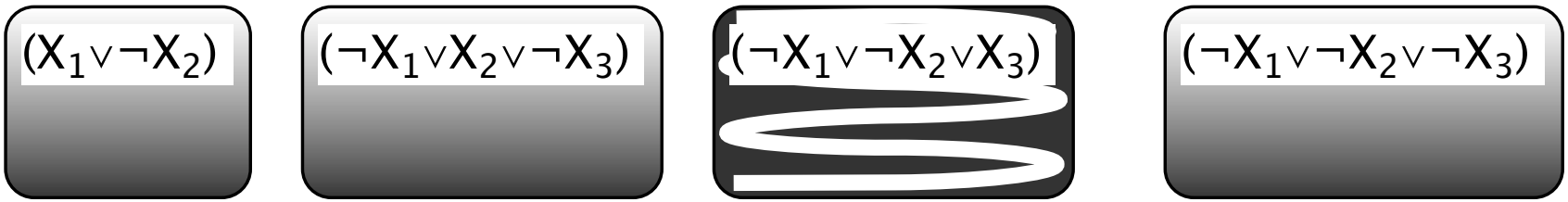
SAT
Esempio



$X_1=1$ $X_2=1$ $X_3=1$

1. Data una possibile soluzione, verificarla è facile.
2. Le possibili soluzioni sono "tante" ($2^{|\mathcal{X}|}$).

SAT
Esempio



$X_1=1$ $X_2=1$ $X_3=0$

1. Data una possibile soluzione, verificarla è facile.
2. Le possibili soluzioni sono "tante" $(2^{|\mathcal{X}|})$.

SAT
Esempio

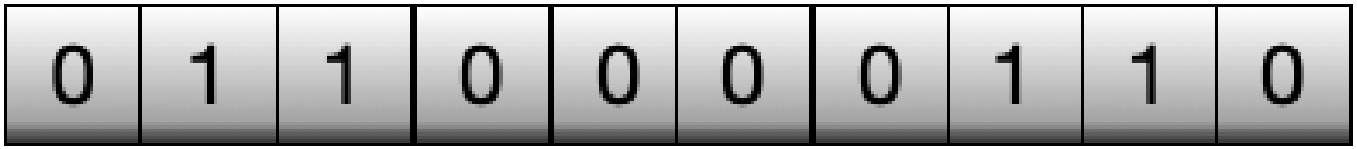
$(X_1 \vee \neg X_2)$ $(\neg X_1 \vee X_2 \vee \neg X_3)$ $(\neg X_1 \vee \neg X_2 \vee X_3)$ $(\neg X_1 \vee \neg X_2 \vee \neg X_3)$

$X_1=1$ $X_3=0$ $X_2=0$

1. Data una possibile soluzione, verificarla è facile.
2. Le possibili soluzioni sono "tante" $(2^{|\mathcal{X}|})$.

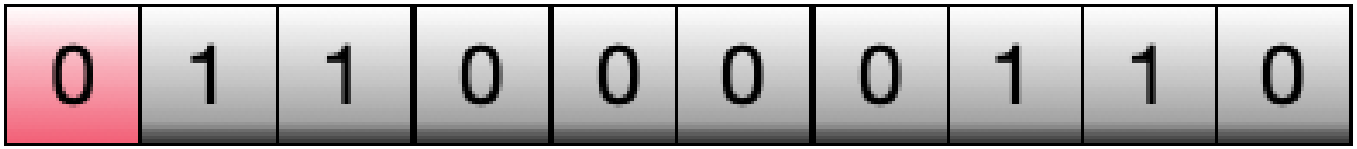
UN PROBLEMA POLINOMIALE: PAL

Problema: stabilire se una stringa/vettore di caratteri (0 e 1) è o meno palindroma (ovvero è la stessa se la leggiamo da sinistra a destra o da destra a sinistra).



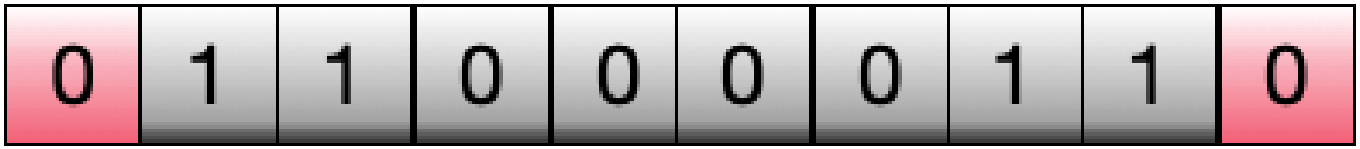
UN PROBLEMA POLINOMIALE: PAL

Problema: stabilire se una stringa/vettore di caratteri (0 e 1) è o meno palindroma (ovvero è la stessa se la leggiamo da sinistra a destra o da destra a sinistra).



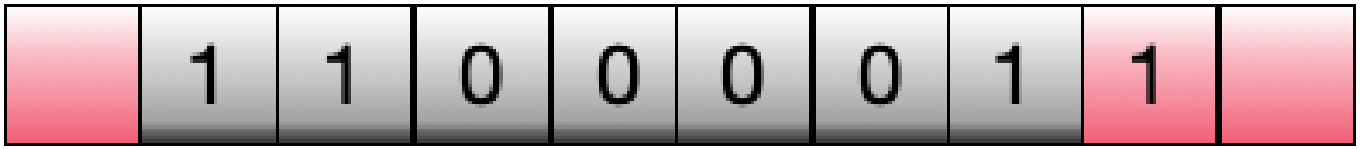
UN PROBLEMA POLINOMIALE: PAL

Problema: stabilire se una stringa/vettore di caratteri (0 e 1) è o meno palindroma (ovvero è la stessa se la leggiamo da sinistra a destra o da destra a sinistra).



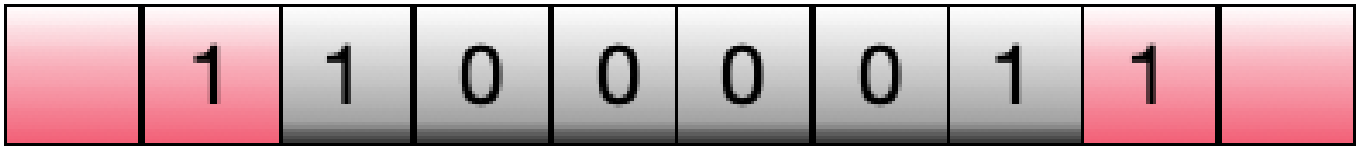
UN PROBLEMA POLINOMIALE: PAL

Problema: stabilire se una stringa/vettore di caratteri (0 e 1) è o meno palindroma (ovvero è la stessa se la leggiamo da sinistra a destra o da destra a sinistra).



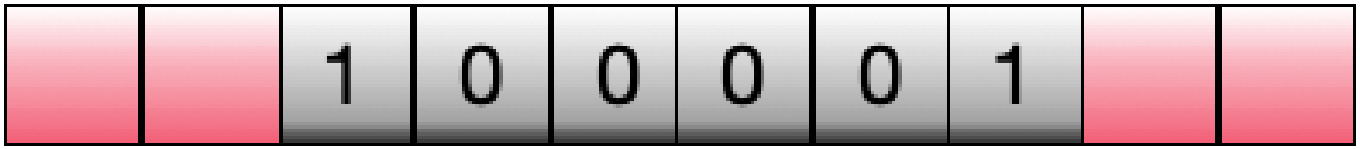
UN PROBLEMA POLINOMIALE: PAL

Problema: stabilire se una stringa/vettore di caratteri (0 e 1) è o meno palindroma (ovvero è la stessa se la leggiamo da sinistra a destra o da destra a sinistra).



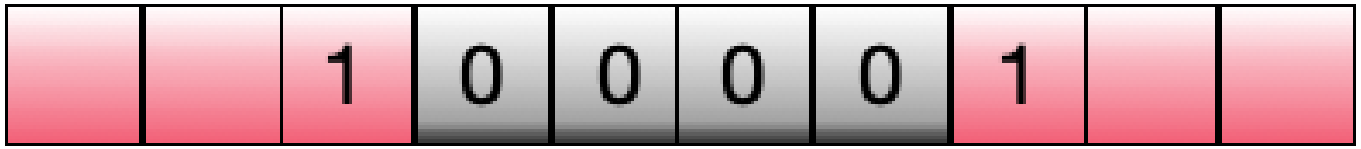
UN PROBLEMA POLINOMIALE: PAL

Problema: stabilire se una stringa/vettore di caratteri (0 e 1) è o meno palindroma (ovvero è la stessa se la leggiamo da sinistra a destra o da destra a sinistra).



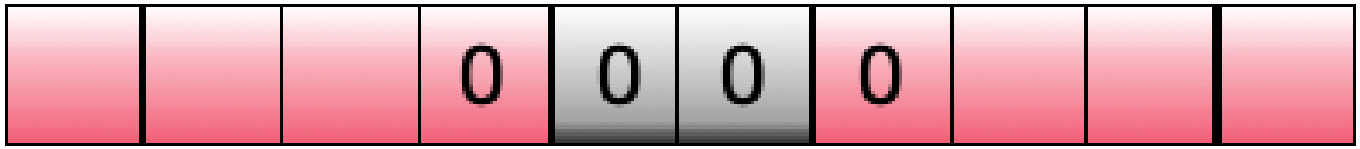
UN PROBLEMA POLINOMIALE: PAL

Problema: stabilire se una stringa/vettore di caratteri (0 e 1) è o meno palindroma (ovvero è la stessa se la leggiamo da sinistra a destra o da destra a sinistra).



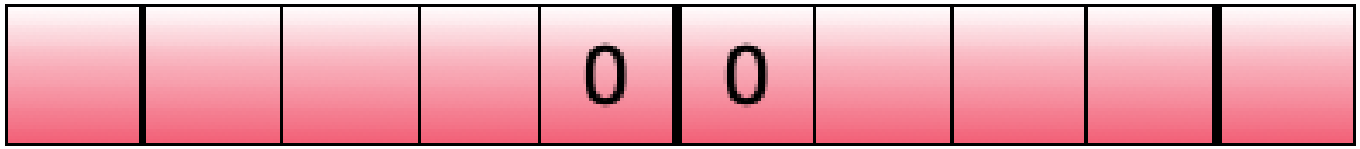
UN PROBLEMA POLINOMIALE: PAL

Problema: stabilire se una stringa/vettore di caratteri (0 e 1) è o meno palindroma (ovvero è la stessa se la leggiamo da sinistra a destra o da destra a sinistra).



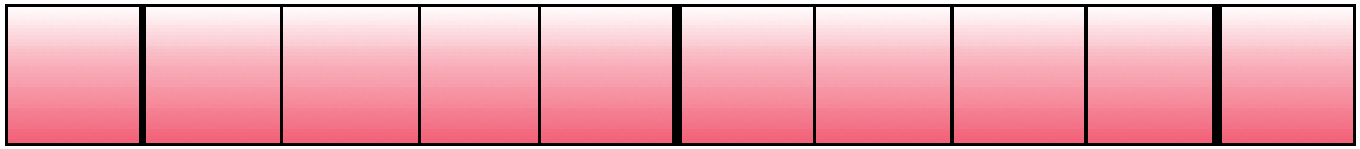
UN PROBLEMA POLINOMIALE: PAL

Problema: stabilire se una stringa/vettore di caratteri (0 e 1) è o meno palindroma (ovvero è la stessa se la leggiamo da sinistra a destra o da destra a sinistra).



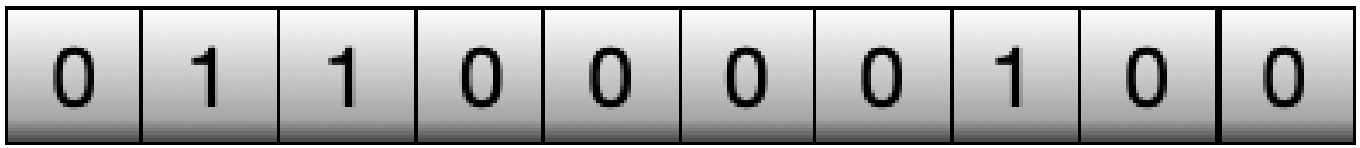
UN PROBLEMA POLINOMIALE: PAL

Problema: stabilire se una stringa/vettore di caratteri (0 e 1) è o meno palindroma (ovvero è la stessa se la leggiamo da sinistra a destra o da destra a sinistra).



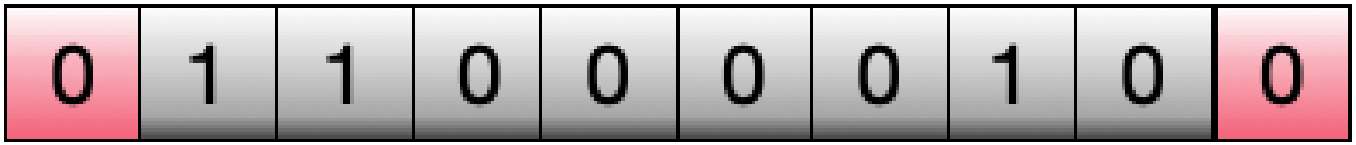
UN PROBLEMA POLINOMIALE: PAL

Problema: stabilire se una stringa/vettore di caratteri (0 e 1) è o meno palindroma (ovvero è la stessa se la leggiamo da sinistra a destra o da destra a sinistra).



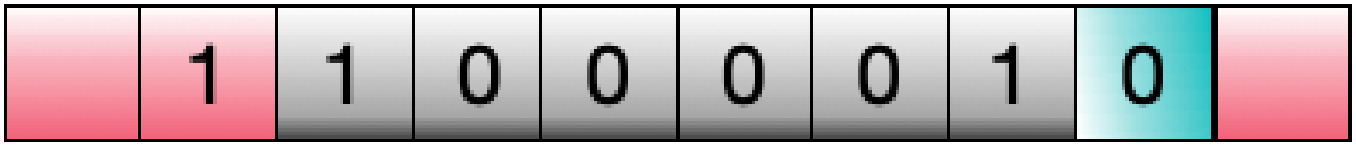
UN PROBLEMA POLINOMIALE: PAL

Problema: stabilire se una stringa/vettore di caratteri (0 e 1) è o meno palindroma (ovvero è la stessa se la leggiamo da sinistra a destra o da destra a sinistra).

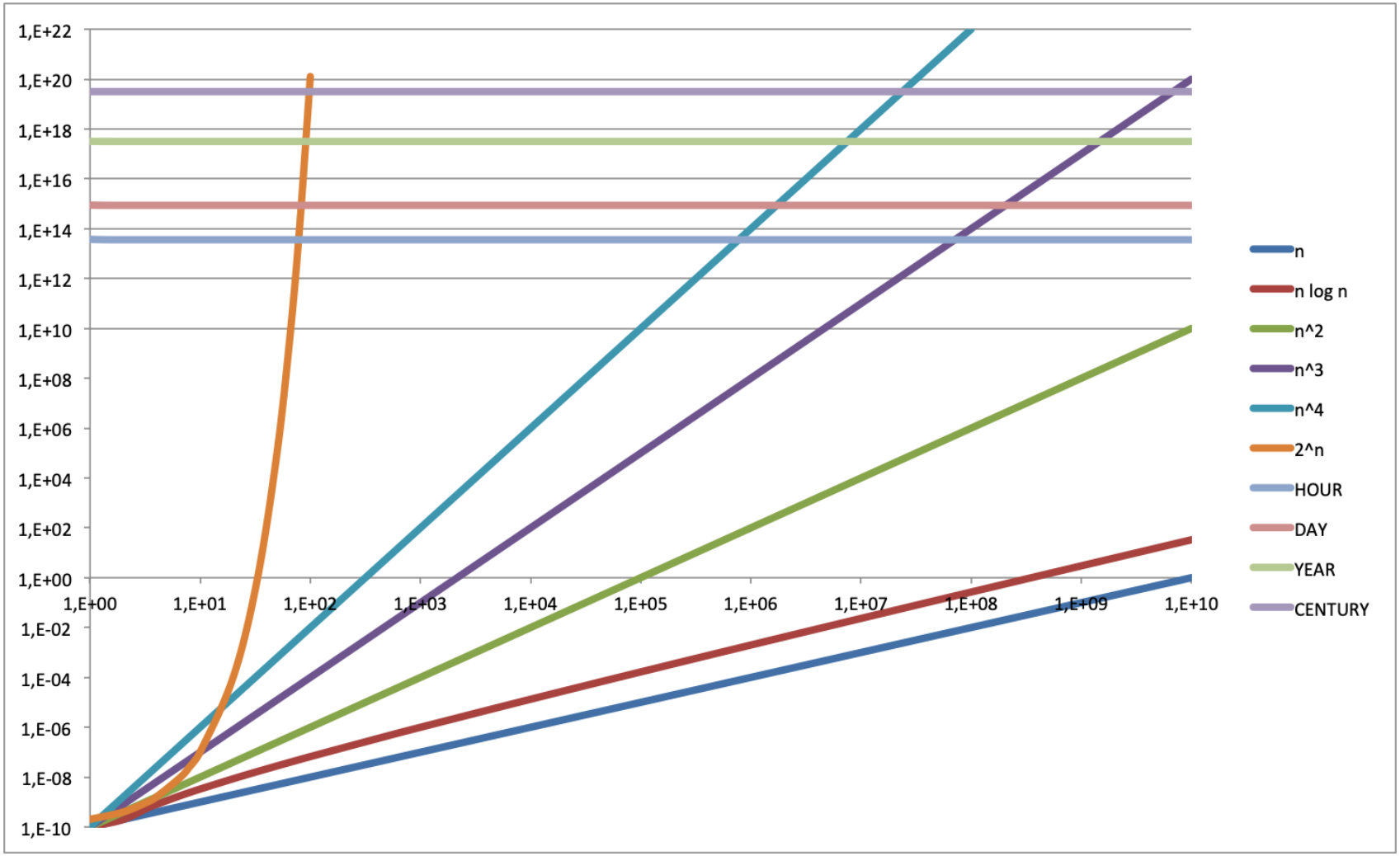


UN PROBLEMA POLINOMIALE: PAL

Problema: stabilire se una stringa/vettore di caratteri (0 e 1) è o meno palindroma (ovvero è la stessa se la leggiamo da sinistra a destra o da destra a sinistra).



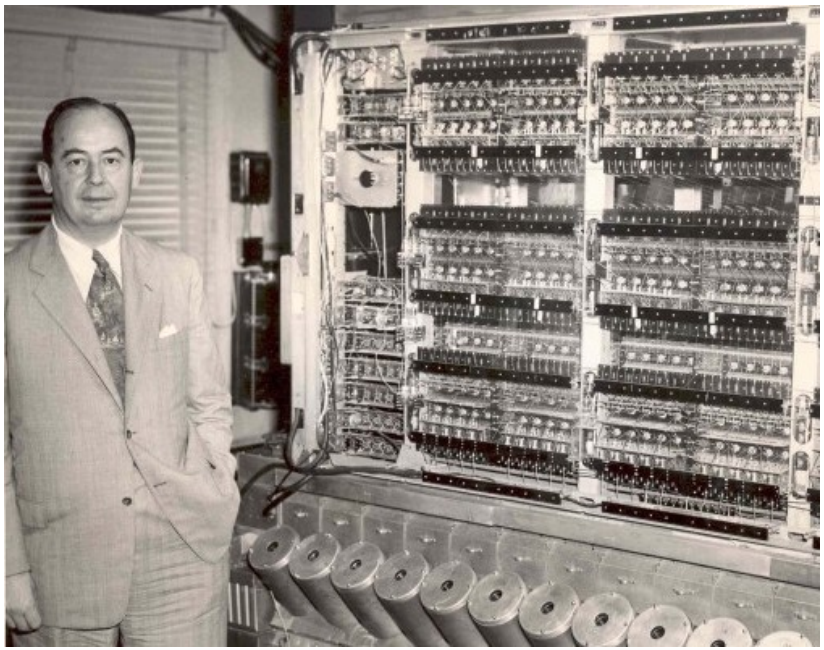
POLINOMIALE VS ESPONENZIALE



La lettera di Gödel a von Neumann

HTTPS://RJLIPTON.WORDPRESS.COM/THE-GDEL-LETTER/

(1906–1978)



(1903–1957)

La lettera di Gödel a von Neumann

HTTPS://RJLIPTON.WORDPRESS.COM/THE-GDEL-LETTER/

Princeton, 20 March 1956

Lieber Herr v. Neumann:

With the greatest sorrow I have learned of your illness. [...] I would like to allow myself to write you about a mathematical problem, of which your opinion would very much interest me: One can obviously easily construct a Turing machine, which for every formula F in first order predicate logic and every natural number n, allows one to decide if there is a proof of F of length n (length = number of symbols). Let $\psi(F, n)$ be the number of steps the machine requires for this and let $\varphi(n) = \max_F \psi(F, n)$. The question is how fast $\varphi(n)$ grows for an optimal machine. [...] If there really were a machine with $\varphi(n) \sim kn$ (or even $\sim kn^2$), this would have consequences of the greatest importance. Namely, it would obviously mean that in spite of the undecidability of the Entscheidungsproblem, the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a machine. [...] Now it seems to me, however, to be completely within the realm of possibility that $\varphi(n)$ grows that slowly. [...]

SAT e Gödel

Senza entrare troppo nella terminologia molto tecnica della lettera, Gödel era ottimista sull'esistenza di un algoritmo polinomiale (addirittura lineare o quadratico) per la risoluzione del problema SAT.

Purtroppo non sappiamo cosa ne pensasse von Neumann (che morí di Cancro poco dopo).

60 anni dopo non sappiamo ancora se Gödel avesse ragione. Nè se avesse torto.

La Complessità computazionale

- Negli anni '70 nasce una teoria matematica (informatica) che cerca di catalogare la difficoltà di un problema sulla base della efficienza di un algoritmo/di un programma al calcolatore che lo risolve
- La teoria non dipende dalla velocità del calcolatore adoperato ma dal numero di passi di computazione necessari.
- Ma, se ci pensate, le differenze tra un calcolatore e l'altro sono poca cosa rispetto alle crescite viste nel diagramma di prima

P VERSUS NP

Vengono definite numerose classi, tra queste:

- P dei problemi che sono risolti da un algoritmo che opera in tempo polinomiale (proporzionale a n, $n \log n$, n^2, n^3, etc.)
  Esempio: data una stringa di 0 e 1 verificare se sia o meno palindroma (n è la sua lunghezza)
- NP dei problemi che, se qualcuno mi fornisce una possibile soluzione, la so verificare in tempo polinomiale
  Esempio: SAT. Il certificato è l'assegnamento $X_1/0, X_2/1, \ldots, X_n/1$ delle variabili (n variabili).
  Ma anche: preparare i turni per le lezioni, i turni di lavoro di una azienda, l'itinerario per portare i regali di Natale comperati on-line, ...
- Cercare UN certificato che verifichi è la parte difficile.
  Tipicamente dobbiamo esplorare uno spazio di ricerca di dimensioni esponenziali rispetto a n.

P VERSUS NP

- E' (abbastanza) evidente che $P \subseteq NP$.
- Ma sarà $P = NP$ o $P \subset NP$?
- Il problema è stabilire quale delle due è vera. Grazie al fondamentale teorema di Cook-Levin:
- Per mostrare che $P = NP$ sarebbe sufficiente scrivere un algoritmo polinomiale che risolva SAT.
- Per mostrare che $P \subset NP$ ($P \neq NP$) bisognerebbe dimostrare che un tale algoritmo non esiste.
- E' un problema importante?

P VERSUS NP

- E' (abbastanza) evidente che $P \subseteq NP$.
- Ma sarà $P = NP$ o $P \subset NP$?
- Il problema è stabilire quale delle due è vera. Grazie al fondamentale teorema di Cook-Levin:
- Per mostrare che $P = NP$ sarebbe sufficiente scrivere un algoritmo polinomiale che risolva SAT.
- Per mostrare che $P \subset NP$ ($P \neq NP$) bisognerebbe dimostrare che un tale algoritmo non esiste.
- E' un problema importante?

I PROBLEMI DEL MILLENNIO

HTTP://WWW.CLAYMATH.ORG/MILLENNIUM-PROBLEMS

1. Yang-Mills and Mass Gap
2. Riemann Hypothesis
3. P vs NP Problem. If it is easy to check that a solution to a problem is correct, is it also easy to solve the problem? This is the essence of the P vs NP question.

   If you prove that $P = NP$ you are able to solve mechanically all the others [Aaronson 2017]
4. Navier-Stokes Equation
5. Hodge Conjecture
6. Poincaré Conjecture (risolta da Grigoriy Perelman nel 2003)
7. Birch and Swinnerton-Dyer Conjecture

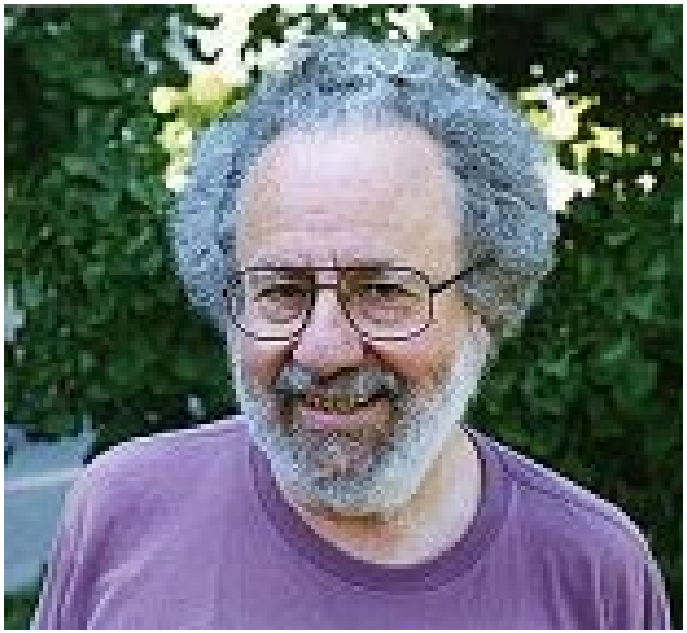
On *P versus NP*

- Abbiamo visto Gödel e Von Neumann
- *Now my general conjecture is as follows: for almost all sufficiently complex types of enciphering . . . the mean key computation length increases exponentially with the length of the key, or in other words, the information content of the key . . . The nature of this conjecture is such that I cannot prove it, even for a special type of ciphers. Nor do I expect it to be proven.* [John Nash, 1955 (Nobel Economia 1994)]
- *P versus NP* — a gift to mathematics from computer science [Steve Smale (Fields 1996, Wolf 2006)]
- *The P versus NP problem deals with the central mystery of computation. The story of the long assault on this problem is our Iliad and our Odyssey; it is the defining myth of our field.* [Eric Allender, 2009]

On *P versus NP*

- Juris Hartmanis (TA 1993). Gödel, von Neumann and the P =? NP Problem (1989).
- Michael Sipser. The History and Status of the P versus NP Question (1992).
- Stephen Cook (TA 1982). The Importance of the P versus NP Question (2003).
- Avi Widgerson. P, NP and mathematics — a computational complexity perspective (2006).
- Eric Allender. A Status Report on the P versus NP Question (2009).
- Scott Aaronson. P $\stackrel{?}{=}$ NP (2017).

SAT SOLVING



Martin Davis (1928)

Hillary Putnam (1926–2016)

Anche se Gödel e Turing hanno dimostrato che è impossibile l'automazione completa della logica del primo ordine, è comunque sensato affrontare l'automazione della ricerca di dimostrazioni di lunghezza finita (la parte "obviously, easily" della lettera di Gödel)

DAVIS PUTNAM —JACM 7(3):201–215 (1960)

Due ingredienti principali:

UNIT RULE Data una clausola

$$\ell_1 \vee \cdots \vee \ell_{n-1} \vee \ell_n$$

tale che l'assegnamento finora calcolato per le variabili rende falsi $\ell_1, \ldots, \ell_{n-1}$ allora ℓ_n va deterministicamente posto a vero.

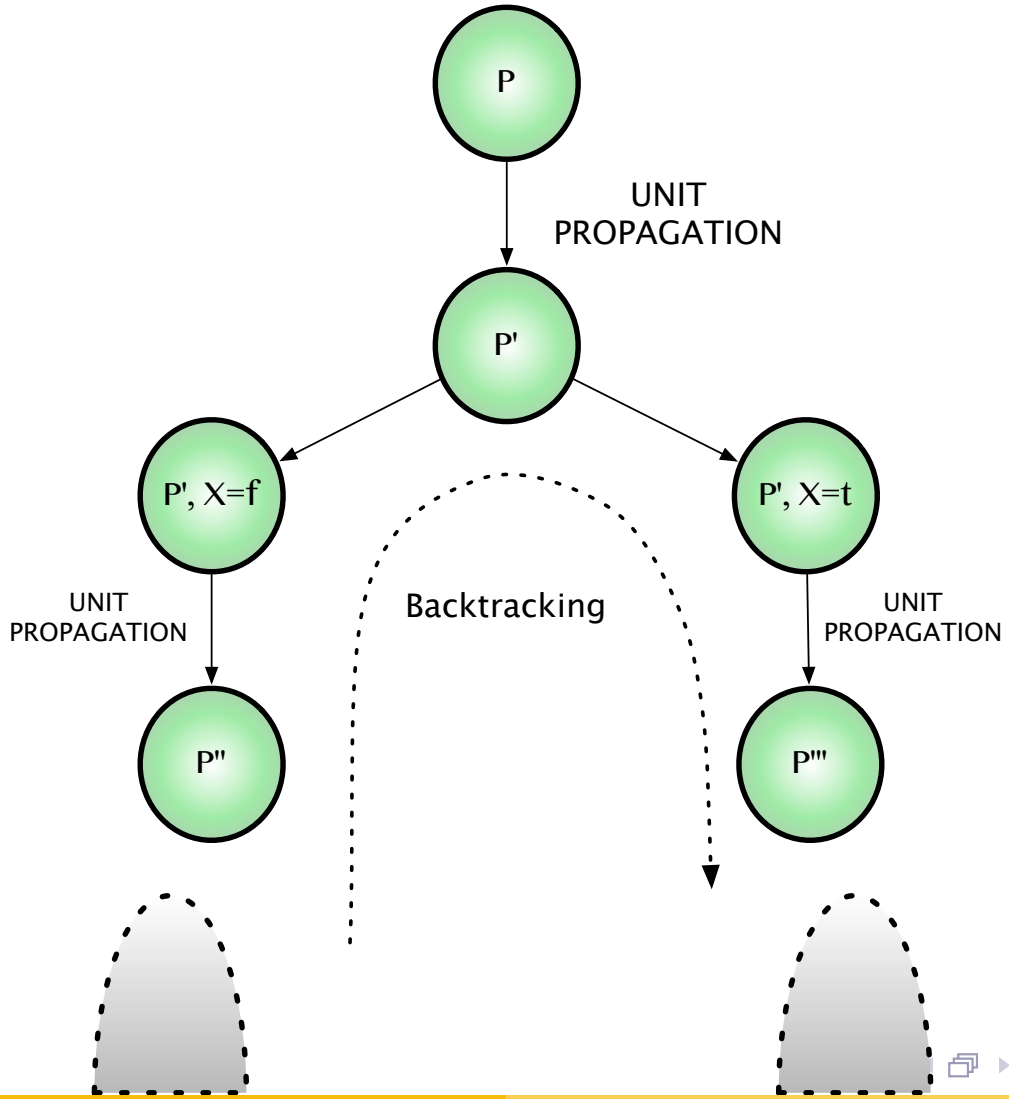
RULE III (RESOLUTION) Riscrivi la formula

$$F = (\vec{a} \vee p) \wedge (\vec{b} \vee \neg p) \wedge R$$

con: $F' = (\vec{a} \vee \vec{b}) \wedge R$

RULE III* (SPLITTING) Scegli una variabile non ancora assegnata x in F. Prova a soddisfare $(x \wedge F)$. Se non ci riesci, allora prova: $(\neg x \wedge F)$
(introdotta da Loveland in [DPLL62])

SAT SOLVING
DPLL (1962)



SAT COMPETITION

- Grazie al risultato di Cook-Levin, ogni problema in NP può essere ridotto (trasformato) in una formula (istanza) di SAT
- Una soluzione efficiente per SAT sarebbe ereditata da tutti i problemi in NP
- Pertanto da DPLL in poi parte la progettazione dei cosiddetti SAT SOLVERS
- Un punto cruciale è la SAT COMPETITION `http://www.satcompetition.org/` organizzata dal 2002.
- Un enorme speed-up avvenne con l'introduzione del conflict-driven clause learning.
- Ogni volta che la ricerca dell'albero finisce in un ramo senza soluzioni (un conflitto) si impara una nuova clausola che migliora il comportamento successivo.

Il formato DIMACS

- Per la SAT competition è stato fissato un formato standard.
- Le istanze sono in un file ASCII (con suffisso ".cnf")
- Le variabili sono rappresentate da numeri interi (eccetto lo zero), con il $-$ si denota la negazione.
- I commenti iniziano con "c", lo "0" termina le clausole
- C'è una linea iniziale iniziante con "p" che memorizza il numero di variabili e clausole.

$$(X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_3)$$

```
c ****************************************
c SAT Encoding of problem above
c ****************************************
p cnf 3 2
1  -2  3 0
-1 -3 0
```

IDEA: RISOLVERE PROBLEMI NP USANDO SAT

- Devo affrontare un problema A che "mi sembra" NP
- Scrivo un programma nel mio linguaggio preferito che data una istanza del problema A la trasforma in una istanza di SAT "equivalente" (nel formato DIMACS)
- In pratica faccio una riduzione $A \leq SAT$
- Uso un SAT solver (dal sito della competizione ne posso scaricare diversi)
- Un grande classico è MiniSAT
- Se il problema da affrontare è NP-completo, non ci pentiremo!

ESEMPIO: SUDOKU

				1				
			2		3			
		4				5		
	6						7	
8				5				9
	1						3	
		5				4		
			7		1			
				9				

ESEMPIO: SUDOKU

- In ogni coppia (r, c) in $\{(1,1),(1,2),\ldots,(9,8),(9,9)\}$ devo mettere uno ed un solo numero da 1 a 9.
- Introduco 9 variabili per ogni cella (r, c): $X_1^{(r,c)}, \ldots, X_9^{(r,c)}$
- Devo dire che almeno una di esse è vera:

$$X_1^{(r,c)} \vee X_2^{(r,c)} \vee X_3^{(r,c)} \vee X_4^{(r,c)} \vee X_5^{(r,c)} \vee X_6^{(r,c)} \vee X_7^{(r,c)} \vee X_8^{(r,c)} \vee X_9^{(r,c)}$$

- Devo dire che mai due di esse sono vere ovvero, per $i = 1, \ldots, 8$ per $j = i + 1, \ldots, 9$:

$$\neg X_i^{(r,c)} \vee \neg X_j^{(r,c)}$$

- Stiamo parlando di 9^3 variabili, dunque di uno spazio di ricerca di dimensione $2^{729} \approx 2.8 \cdot 10^{219}$

ESEMPIO: SUDOKU

- In ogni riga r per ogni valore k dall'1 al 9 c'è almeno una volta
- Devo dire che almeno una di esse è vera:

$$X_k^{(r,1)} \vee X_k^{(r,2)} \vee X_k^{(r,3)} \vee X_k^{(r,4)} \vee X_k^{(r,5)} \vee X_k^{(r,6)} \vee X_k^{(r,7)} \vee X_k^{(r,8)} \vee X_k^{(r,9)}$$

- Ma non due volte: Devo dire che mai due di esse sono vere ovvero, per $i = 1, \ldots, 8$ per $j = i + 1, \ldots, 9$:

$$\neg X_k^{(r,i)} \vee \neg X_k^{(r,j)}$$

- Analogamente per le colonne (ve la lascio)

ESEMPIO: SUDOKU

0	0	0	1	1	1	2	2	2
0	0	0	1	1	1	2	2	2
0	0	0	1	1	1	2	2	2
3	3	3	4	4	4	5	5	5
3	3	3	4	4	4	5	5	5
3	3	3	4	4	4	5	5	5
6	6	6	7	7	7	8	8	8
6	6	6	7	7	7	8	8	8
6	6	6	7	7	7	8	8	8

Identifico i sottoquadrati

- Se le righe r vanno da 0 a 8 e le colonne c vanno da 0 a 8
- La cella (r, c) sta nel quadrato $3 * (r \,\texttt{div}\, 3) + c \,\texttt{div}\, 3$.
- Con questa idea si mettono i vincoli (analoghi ai precedenti) sulle variabili in ogni sottoquadrato.
  Se r e c vanno da 1 a 9 (anziché da 0 a 8) basta mettere qualche "+1" e "-1" :)
- Quanto visto non dipende dalla specifica istanza!

ESEMPIO: SUDOKU

				1				
			2		3			
		4				5		
	6						7	
8				5				9
	1						3	
		5				4		
			7		1			
				9				

Devo esplicitare i fatti noti

- $X_1^{(1,5)}$
- $X_2^{(2,4)}$ e $X_3^{(2,6)}$
- ...
- $X_9^{(9,5)}$

ESEMPIO: SUDOKU

A questo punto dobbiamo solo trovare un mapping sensato tra

$$X_k^{(r,c)}$$

e un numero da 1 a 729 (per il DIMACS format)

Diamo un occhiata a un codice in C e alla sua esecuzione.

Certo, tutto molto bello, ma scrivere/modificare una codifica in SAT può essere un incubo.

ESEMPIO: SUDOKU

A questo punto dobbiamo solo trovare un mapping sensato tra

$$X_k^{(r,c)}$$

e un numero da 1 a 729 (per il DIMACS format)

Diamo un occhiata a un codice in C e alla sua esecuzione.

Certo, tutto molto bello, ma scrivere/modificare una codifica in SAT può essere un incubo.

ASP: UN LINGUAGGIO LOGICO PER NP

```
coord(1..9).
val(1..9).
% Assegno una coppia (X,Y) ad un sottoquadrato - e viceversa
square(I,X,Y) :- coord(X),coord(Y),coord(I),
   I == (X-1) / 3 + 3*((Y-1) / 3) + 1.
% Per ogni cella si assegna esattamente un valore
1 { x(X,Y,N) : val(N) } 1   :- coord(X), coord(Y).
% Ogni valore viene usato esattamente una volta in una colonna
1 { x(X,Y,N) : coord(X) } 1 :- coord(Y), val(N).
% Ogni valore viene usato esattamente una volta in una riga
1 { x(X,Y,N) : coord(Y) } 1 :- coord(X), val(N).
% Ogni valore viene usato esattamente una volta in un sottoquadr
1 { x(X,Y,N) : square(I,X,Y) } 1  :- val(N), coord(I).
% istanza
x(1,5,1).  x(2,4,2). x(2,6,3).  ...
```

Conclusioni

- Importanza teorica e pratica del problema SAT:
- Il problema P vs NP vi aspetta!
- Se dovete affrontare un problema in NP (che non vi sembra facile) NON scrivete una soluzione ad-hoc ma trasformatelo in istanze di SAT (o ASP)
- I SAT solver (ma anche gli ASP solvers, e altre tecniche che si imparano ad informatica) risolvono il vostro problema meglio di quello che potreste fare in diversi mesi forse anni di codifica diretta.
- Inoltre, se trovaste un algoritmo polinomiale per SAT (o se dimostraste che un tale algoritmo non può esistere) vincereste 1 milione di dollari, oltre agli onori che ne seguirebbero . . .