Tommaso Castellani

Risolvere i problemi difficili

Sudoku, commessi viaggiatori e altre storie

Chiavi di lettura a cura di
Federico Tibone e Lisa Vozza

CHIAVI DI LETTURA ZANICHELLI

indice

Introduzione	5
1. Problemi difficili	11
2. Computer e complessità	43
3. La fisica dei sistemi complessi	75
4. La fisica dei problemi difficili	117
Epilogo: Comportamenti di gruppo	145
Per saperne di più	152
6 miti da sfatare	154
Forse non sapevi che…	156
Indice analitico	158

Introduzione

Questo libro tocca argomenti apparentemente molto diversi tra loro, come il Sudoku, i materiali vetrosi, la ricerca di numeri primi sempre più grandi, i pagamenti sicuri on-line, l'ebollizione dell'acqua.

Vi domanderete che cosa abbiano in comune. Per il momento, diremo soltanto che gettano luce sulla relazione tra l'informatica computazionale e la fisica dei problemi complessi. Ma per scoprire che cosa ciò significhi davvero, non c'è altra soluzione che procedere nella lettura. Diamo allora uno sguardo al percorso che seguiremo nel testo.

Per prima cosa incontreremo i cosiddetti problemi di ottimizzazione, in cui di solito si deve rendere massima o minima una quantità sottoposta a certi vincoli. Tra gli esempi più antichi c'è il «problema di Erone», che richiede di determinare il cammino più breve tra due punti A e B che tocca una retta data. Sebbene questo problema sia stato risolto elegantemente da Erone ben duemila anni fa – il cammino più breve è composto dai due segmenti che formano angoli uguali con la retta – la soluzione è tutt'altro che banale da dimostrare, e ha dato filo da torcere ai liceali nel compito di matematica dell'Esame di Stato del 2012.

Un altro esempio antico è il «problema di Didone», legato alla leggenda della fondazione di Cartagine. Si narra infatti che Didone, giunta sui lidi dell'Africa settentrionale, desiderasse acquistare un terreno per costruire una città. Il re del luogo, per prenderla in giro, le offrì «tanta terra quanta ne può contenere una pelle di bue». L'astuta Didone, però, tagliò la pelle di bue in modo da ricavarne una sottilissima striscia, con la quale contornò una porzione di terreno di dimensioni molto più grandi del lembo di pelle. Della leggenda si accenna anche nel primo libro dell'Eneide (365–368):

> Devenere locos ubi nunc ingentia cernis
> moenia surgentemque novae Karthaginis arcem,
> mercatique solum, facti de nomine Byrsam,
> taurino quantum possent circumdare tergo.*

La domanda è: una volta avuta la geniale idea di ricavare dalla pelle una striscia, quale forma deve dare Didone alla striscia per prendersi la maggior quantità possibile di terreno? A quanto pare, la speculazione edilizia era una questione già attuale.

In termini matematici: tra le figure geometriche che hanno un dato perimetro, qual è quella con l'area massima? In questo caso la risposta giusta è quella intuitiva: si tratta del cerchio. Ma per averne una dimostrazione rigorosa si è dovuto attendere l'Ottocento.

* Giunsero nei luoghi dove ora vedrai sorgere le enormi mura e la rocca della nuova Cartagine, chiamata *Birsa* [«pelle di bue»] perché fu acquistato tanto terreno quanto se ne potesse circondare con una pelle di toro.

In altre varianti del problema, Didone – evidentemente consapevole delle potenzialità turistiche di quella che oggi è la Tunisia – non vuole rinunciare a una bella porzione di spiaggia: in tal caso traccerà un semicerchio di terraferma che termina con il diametro in riva al mare.

Di problemi legati alla massimizzazione di aree e volumi ce ne sono a bizzeffe, alcuni con evidenti applicazioni pratiche: nel Seicento, per esempio, Keplero si interrogò sulla forma ottimale da dare a una botte di vino.

I problemi di ottimizzazione di tipo geometrico, come quelli finora citati, divennero facilmente risolvibili quando Newton e Leibniz svilupparono la matematica che permette di trovare i massimi e i minimi delle funzioni.

Nell'Ottocento però sono stati formulati anche problemi di *ottimizzazione combinatoria*, che sono più difficili da descrivere matematicamente. Tra i più famosi c'è il «problema del commesso viaggiatore», che chiede quale sia il percorso più breve per visitare un dato numero di città. Sebbene possa sembrare soltanto una versione un poco più elaborata del problema di Erone, il problema del commesso viaggiatore si rivela molto difficile da studiare. Addirittura, per motivi che vedremo nel corso di questo libro, se il numero delle città è molto alto il problema è ancora oggi *impossibile da risolvere*, perfino con l'ausilio dei più potenti computer.

Il problema del commesso viaggiatore rientra infatti tra quelli che chiameremo *problemi difficili*, una categoria di problemi che costituisce una delle sfide aperte dell'*informatica computazionale*, cioè la branca dell'informatica che si occupa in particolare di fare calcoli per risolvere nel modo più efficiente possibile i problemi.

Nel primo capitolo ne presenteremo alcuni e cominceremo a riflettere sulla natura della loro difficoltà. Nel secondo capitolo esamineremo più a fondo la relazione tra i problemi difficili e il computer, passando in rassegna alcune applicazioni concrete.

Di fronte all'impossibilità di affrontare i problemi difficili, ci verrà in aiuto in maniera inattesa la *fisica dei sistemi complessi*.

Di che cosa si tratta? Per introdurre il tema possiamo raccontare una barzelletta che circola tra i fisici. Un allevatore di mucche da latte è disperato perché, da qualche tempo, i suoi animali producono meno latte di prima. Decide di affrontare il problema in maniera scientifica e convoca un gruppo di fisici. Gli scienziati si mettono al lavoro e dopo qualche tempo l'allevatore chiede loro se sono giunti a qualche conclusione: «Siamo a buon punto» rispondono. «Per ora sappiamo risolvere il problema nel caso di una mucca di forma sferica con distribuzione di latte uniforme».

Questa storiella fa ridere molto i fisici, soprattutto i fisici teorici, perché ironizza su una caratteristica fondamentale dei modelli fisici: la semplicità.

Anche nella fisica che studiamo a scuola troviamo il moto di corpi puntiformi (cioè senza dimensioni), leggi della dinamica in un mondo senza attrito, gas perfetti, forze elettriche tra cariche ancora una volta puntiformi, e via dicendo. Il mondo della fisica sembra così lontano dal mondo reale, fatto di corpi non puntiformi, mucche non sferiche e gas non perfetti!

Nel libro cercheremo di spiegare come il fisico in effetti produca *modelli* la cui corrispondenza con la realtà non è così diretta come potrebbe sembrare. I modelli sono semplici non perché si ipotizzi che la realtà sia semplice, ma perché si sceglie di *isolarne alcuni aspetti* e studiarne pochi alla volta.

Con il progredire della fisica i modelli sono diventati più sofisticati, includendo un numero sempre maggiore di fattori; ma non per questo sono più realistici. Quando si sono costruiti modelli di sistemi collettivi composti da un gran numero di elementi semplici, però, si è osservato spesso l'emergere di fenomeni particolarmente elaborati, malgrado l'estrema semplicità dei costituenti fondamentali: per questo si parla di «sistemi complessi».

Anche nello studio dei sistemi collettivi, naturalmente, i fisici hanno cominciato dai casi più facili. Si sono concentrati così su sistemi fortemente omogenei, o al più con asimmetrie molto semplici.

Negli ultimi decenni, tuttavia, è nata una branca della fisica che ha cominciato a occuparsi dei sistemi detti «disordinati», in cui non è evidente alcuna forma di simmetria né è possibile identificare asimmetrie semplici.

Nel terzo capitolo mostreremo alcuni di questi modelli fisici, che sono astratti e apparentemente lontani della realtà. Nel quarto capitolo infine vedremo come, sorprendentemente, i modelli di sistemi disordinati hanno trovato applicazione in un campo completamente diverso da quello in cui sono stati sviluppati.

Questo campo è proprio l'informatica computazionale, e qui il cerchio di chiude: la fisica dei sistemi disordinati permette di scoprire la vera natura della difficoltà dei «problemi difficili», aprendo la strada a nuovi algoritmi per la loro risoluzione.

L'alleanza tra i fisici statistici e gli informatici computazionali rappresenta, nella scienza moderna fortemente specializzata, un raro esempio di genuina interdisciplinarità.

Problemi difficili

Nel 1883 un non meglio identificato N. Claus de Siam, mandarino del collegio di Li-Sou-Stian, mise in vendita in Francia «un vero rompicapo annamita proveniente dal Tonchino» (la Catena Annamita è una catena montuosa, il Tonchino è la regione del Vietnam in cui si trova). Il nome del gioco era *La torre di Hanoi*.

La torre di Hanoi

La figura 1 mostra una versione del rompicapo con 8 dischi. In tre paletti sono infilati dischi di diametri diversi. Si parte mettendo in pila tutti i dischi sul primo paletto, ordinandoli verso l'alto dal più grande al più piccolo.

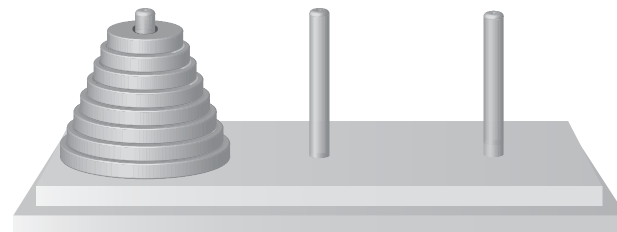

Figura 1. La torre di Hanoi.

Lo scopo del gioco è trasferire tutti i dischi su un altro paletto, muovendo un solo disco alla volta e facendo in modo che un disco più grande non si trovi *mai* sopra un disco più piccolo.

Si può elaborare una strategia vincente? In effetti il misterioso mandarino altri non era che il matematico francese Édouard Lucas (anagramma di Claus), che insegnava a Parigi al liceo Saint-Louis (anagramma di Li-Sou-Stian) e si occupava di teoria dei numeri. Il gioco l'aveva inventato lui, e infatti risulta molto interessante dal punto di vista matematico. Vediamo come lo si risolve.

I matematici spesso sono considerati tipi bizzarri, ma soltanto da chi non conosce alcuni trucchi basilari delle dimostrazioni matematiche. Uno di questi trucchi è partire, per studiare un problema, dal caso più semplice.

Nel nostro problema il caso più semplice è quello di un solo disco. Supponiamo che il paletto di partenza sia il primo a sinistra e quello d'arrivo il primo a destra, e chiamiamo «mossa 0» la situazione di partenza.

La figura 2 schematizza la situazione. Contiamo ora quante mosse sono necessarie per risolvere il gioco. Chiaramente, basta una mossa sola.

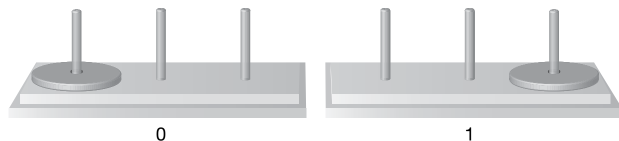

Figura 2. La soluzione della torre di Hanoi con 1 disco.

Sarete perplessi: davvero c'era bisogno di pensarci, e di disegnare addirittura una figura?

Proprio per il fatto di passare del tempo a disegnare figure come questa, i matematici vengono irrisi dai loro colleghi fisici o ingegneri, che si ritengono impegnati in faccende ben più utili.

Ma il matematico è ben contento di aver stabilito un punto di partenza: «Per risolvere il gioco con 1 disco occorre 1 mossa». E vedremo che ha ragione di compiacersi di questo risultato.

Passiamo al grado successivo di complessità, il gioco con due dischi.

In questo caso, come mostra la sequenza della figura 3, per spostarli tutti sul paletto di destra occorrono almeno tre mosse.

Il bravo matematico appunta sul suo foglietto: «1 disco 1 mossa, 2 dischi 3 mosse».

Che cosa succederà con tre dischi? Con una certa sorpresa, dalla figura 4 della prossima pagina scopriamo che sono necessarie ben sette mosse.

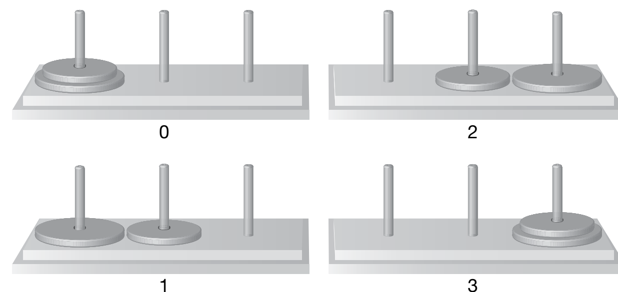

Figura 3. La soluzione della torre di Hanoi con 2 dischi.

A questo punto il matematico ha tutto chiaro, e afferma: «Per risolvere il problema con *n* dischi occorrono almeno $2^n - 1$ mosse».

In altre parole: ci vuole 1 mossa per $n = 1$, ci vogliono 3 mosse per $n = 2$, e 7 mosse per $n = 3$.

Per dimostrare la formula generale, il matematico procederà con il metodo chiamato *induzione*: si aggiunge ogni volta un disco e si constata che, se la legge valeva nel caso precedente, vale anche nel successivo.

Ma torniamo per un momento al problema a 3 dischi e ricapitoliamo i passi da fare per risolvere la torre di Hanoi.

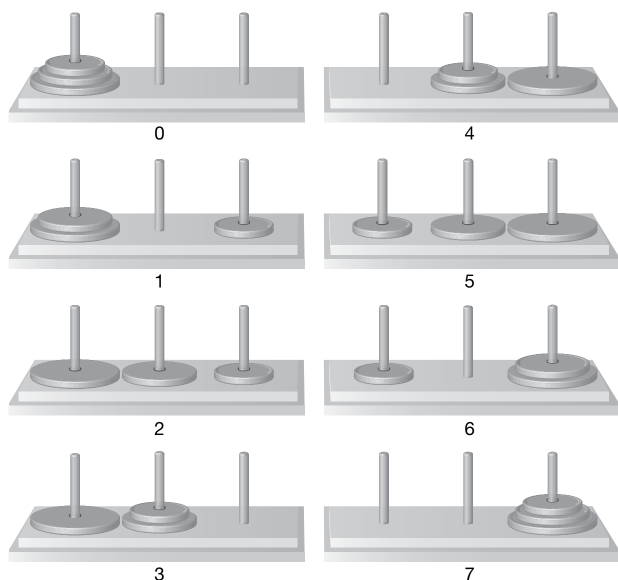

Figura 4. La soluzione della torre di Hanoi con 3 dischi.

Ora suddividiamo però i passi in un modo diverso:

a. spostare tutti i dischi tranne il più grande (in questo caso due dischi) dal primo al secondo paletto (mosse da 1 a 3 nella nostra figura);

b. spostare il disco più grande sul terzo paletto (mossa 4);

c. spostare tutti i dischi tranne il più grande dal secondo al terzo paletto (mosse da 5 a 7).

In altre parole, occorre spostare *due volte* due dischi da un paletto all'altro, e in più fare una mossa aggiuntiva per spostare il disco grande.

Potete vedere che questo vale in generale: per spostare *n* dischi bisogna muovere due volte (n–1) dischi, e poi fare un'altra mossa per spostare il disco più grande.

Abbiamo dunque trovato un procedimento generale per risolvere una torre di Hanoi con *n* qualunque: abbiamo cioè stabilito un *algoritmo* per risolvere il problema.

L'algoritmo produce una sorta di scatole cinesi: nei passi **a** e **c** è contenuta un'istruzione per eseguire la quale abbiamo bisogno di nuovo dei tre passi **a**, **b** e **c**, e così via finché quel «tutti i dischi tranne il più grande» non significhi «un solo disco».

L'affermazione banale che per risolvere il problema con un disco è necessaria una mossa diventa così il pilastro su cui si regge tutto il resto, come accade in tutte le dimostrazioni per induzione.

Vediamo i dettagli della dimostrazione. L'obiettivo è dimostrare che la legge vale per un certo numero n di dischi, ammettendo di aver già dimostrato che vale per il precedente numero di dischi (n–1). Scriviamo in formu-

le la nostra affermazione che per spostare n dischi bisogna muovere due volte $(n-1)$ dischi e fare un'altra mossa per spostare il disco più grande: detto M_n il numero di mosse necessario a spostare n dischi, possiamo scrivere

$$M_n = 2\ M_{n-1} + 1.$$

Supponendo ora che la nostra regola sia valida per $(n-1)$ dischi, cioè che per spostare $(n-1)$ dischi occorrano $(2^{n-1}-1)$ mosse, possiamo riscrivere questo numero come

$$M_n = 2\ (2^{n-1}-1)+1.$$

Se conoscete le regole delle potenze non avrete difficoltà ad accorgervi che questo numero vale 2^n-1. Ricapitolando, abbiamo dimostrato che se la legge vale per $(n-1)$ dischi, allora vale anche per n dischi. Se siamo sicuri che vale per il più piccolo n possibile, cioè 1, saremo sicuri che vale per tutti gli altri valori di n fino all'infinito. Ebbene, per $n = 1$ si ha che 2^n-1 vale 1, e abbiamo visto che per spostare un disco è necessaria proprio 1 mossa: dunque per il più piccolo n possibile la legge è valida, e la nostra dimostrazione è conclusa.

In quanto tempo si risolve la torre di Hanoi?

Parrebbe che abbiamo detto tutto: dovremmo ormai poter risolvere torri di Hanoi con un numero qualsiasi di dischi. Il lavoro difficile è stato trovare la strategia, ora si tratta soltanto di fare le mosse, meccanicamente, una dopo l'altra. Che c'è di interessante ancora?

Se vi procurate una torre di Hanoi a otto dischi, come quella della figura 1 all'inizio del capitolo, e provate a risolverla seguendo il procedimento che abbiamo descritto, vi renderete conto che ci vuole

una certa pazienza: occorre un bel po' di tempo per fare tutte le mosse.

La questione è: *quanto* tempo? Da questa semplice domanda – apparentemente innocua – scaturisce tutto il seguito di questo libro.

Bisogna sapere infatti che il gioco uscì accompagnato da una leggenda, forse inventata da Lucas stesso, che narra di un tempio in cui i monaci stanno lavorando a una torre di Hanoi con 64 dischi: una volta che li avranno spostati tutti, il mondo finirà.

La nostra domanda può essere allora riformulata così: se la leggenda fosse vera, quanto ci resterebbe ancora da vivere?

Facciamo dunque una stima del tempo necessario per risolvere il nostro rompicapo. Supponiamo di aver imparato talmente bene l'algoritmo da riuscire a fare tutte le mosse una dopo l'altra, senza fermarci a pensare.

Possiamo stimare di riuscire a fare, nel migliore dei casi, una mossa al secondo: più di così, è fisicamente impossibile. Con la torre a 8 dischi, avremo bisogno allora di 2^8-1 secondi, cioè 255 secondi, cioè poco più di 4 minuti. Questo è il tempo minimo di risoluzione per 8 dischi. Con 10 dischi ci serviranno $2^{10}-1$ secondi, cioè 1023 secondi, circa 17 minuti. Con 15 dischi il tempo necessario sale a 32 767 secondi, più di 9 ore.

Stiamo parlando sempre del tempo *minimo*: per spostare questi 15 dischi da un paletto all'altro occorre cioè fare per più di nove ore freneticamente una mossa al secondo.

Come vedete, il nostro tempo sta salendo molto, molto rapidamente. Con 20 dischi, risultano necessari 12 giorni, lavorando senza sosta anche di notte. Con 30 dischi ci vorrebbero più di 34 anni. Con 40 dischi, quasi trentacinquemila anni.

Per spostare i 64 dischi sarebbero necessari 585 *miliardi di anni*. Possiamo dunque sfatare la leggenda: a quanto ci dicono gli astrofisici, il Sole si spegnerà tra soli 5 miliardi di anni, o giù di lì. Anche nell'assurda ipotesi che la vita umana sulla Terra resista intatta fino ad allora, i poveri monaci saranno *appena all'inizio* del loro lavoro!

Che cosa sta succedendo in questo problema? Succede che il tempo di risoluzione aumenta in modo *esponenziale*. Siamo abituati a usare questa parola nel linguaggio quotidiano: se cercate con Google «aumento esponenziale» vedrete che si parla della crescita di malattie, reati, suicidi (le cose brutte aumentano sempre in maniera esponenziale).

In realtà, per fortuna, quasi mai questi aumenti sono davvero esponenziali, cioè descrivibili con una funzione in cui la variabile compare all'esponente.

Nel nostro gioco, invece, il tempo di risoluzione del problema è proprio una funzione esponenziale nel numero n dei dischi, giacché è dell'ordine di 2^n. L'aumento esponenziale (quello vero) è velocissimo, tanto che con soli 64 dischi siamo già arrivati a un tempo ben più lungo della vita dell'universo.

I problemi il cui tempo di risoluzione aumenta esponenzialmente con il crescere delle variabili sono piuttosto scoraggianti.

Potremmo pensare di risolvere una versione simulata al computer della nostra torre di Hanoi a 64 dischi con il più veloce computer attualmente disponibile, che si stima sia in grado di elaborare all'incirca un milione di miliardi di operazioni al secondo. Effettivamente, con una macchina del genere potremmo effettuare tutte le mosse necessarie in sole 5 ore. Ma se la torre avesse 100 dischi, anche un computer con queste prestazioni impiegherebbe 40 milioni di anni per compiere tutte le mosse.

L'aumento esponenziale fa sì che, per quanto ci sforziamo di velocizzare la nostra tecnica, basta un piccolo aumento del numero di variabili per vanificare ogni progresso e rendere necessario un tempo molto più lungo di quello che abbiamo a disposizione.

Il Sudoku

Un rompicapo divenuto molto popolare negli ultimi anni è il *Sudoku*.

Come probabilmente saprete, questo gioco si basa su un quadrato suddiviso in 81 caselle, raggruppate in quadrati da 9 caselle.

Molte caselle sono vuote, mentre in alcune appaiono cifre comprese tra 1 e 9.

Lo scopo del gioco è riempire le caselle vuote in modo tale che (a) in ogni quadrato piccolo di 9 caselle compaiano tutte e 9 le cifre, e (b) in ogni riga e in ogni colonna del quadrato grande compaiano tutte e 9 le cifre.

Provate a risolvere il Sudoku della figura 5a!

Poi, quando vorrete, controllate la soluzione nella figura 5b. Noterete che le cifre rispettano i requisiti di non ripetersi mai, né all'interno dei quadrati piccoli né su righe e colonne.

Come facciamo a essere certi che questa sia l'*unica* soluzione? Se lo schema di partenza avesse soltanto poche caselle con cifre già inserite, è facile intuire che si potrebbero trovare molte diverse soluzioni. Qual è il numero «giusto» di caselle inizialmente piene?

Trovare una risposta generale a questa domanda è un problema complicato, che i matematici stanno ancora studiando. In un lavoro del gennaio 2012 McGuire, Tugemann e Civario hanno dimostrato che il minimo numero di caselle piene per avere un Sudoku a soluzione unica è 17.

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

Figura 5a. Uno schema Sudoku non molto difficile.

Fare affermazioni generali sul numero di soluzioni è molto difficile; decisamente più facile è generare particolari schemi Sudoku che rispondano al requisito di avere una soluzione unica (e tutti i Sudoku che compaiono su giornali e riviste sono di questo tipo), ma qui non ci occuperemo di questo problema.

Domandiamoci invece: qual è la strategia migliore per risolvere il rompicapo?

Qui le cose vanno in maniera un po' diversa che per la torre di Hanoi, dove seguendo alcune regole fisse potevamo spostare i dischi da un paletto all'altro senza dover pensare a ciò che facevamo.

Osserviamo lo schema della figura 5a: al centro del quadrato c'è una casella vuota. Poiché nel quadrato centrale sono presenti tutte le cifre tranne l'8, è una *mossa obbligata* inserire la cifra 8 in questa casella.

Ci sono altre mosse obbligate? Sì: esaminando lo schema notiamo che nella quarta riga mancano soltanto le cifre 7 e 9, e l'ultima casella a destra in quella riga si trova in una colonna in cui, nel quadrato superiore, già compare la cifra 7: perciò è chiaro che un'altra mossa obbligata è mettere il 9 in questa casella.

A questo punto, di conseguenza, un'ulteriore mossa obbligata è mettere il 7 nella seconda casella della quarta riga, che è l'ultima rimasta libera. Come vedete, una mossa obbligata può produrne altre.

Potrebbe accadere che di mossa obbligata in mossa obbligata ci si ritrovi ad aver riempito tutto lo schema: il Sudoku sarebbe risolto. Ma questa situazione *non è* il caso generale. Un Sudoku del genere sarebbe classificato come mediamente facile.

In un Sudoku «normale», invece, a un certo punto le mosse obbligate finiscono: ci sono caselle bianche con *più di una possibilità* per le cifre da inserire. Come ce la caviamo allora?

Posiamo la penna e prendiamo matita e gomma. Un po' come facciamo per le parole crociate senza schema, o per altri giochi analoghi, *andiamo per tentativi*. Proviamo cioè a inserire nella casella uno dei possibili numeri, e vediamo che cosa succede andando avanti.

Naturalmente dobbiamo essere pronti a cancellare tutte le mosse fatte da questo punto in poi, se ci accorgiamo che questa strada conduce a un vicolo cieco, cioè a una casella che non possiamo riempire con nessuna cifra senza violare le regole. In questo caso dovremo tornare indietro e fare un'ipotesi diversa per una delle caselle che avevamo riempito a matita.

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

Figura 5b. La soluzione del Sudoku della figura 5a.

Il Sudoku appartiene dunque alla categoria dei giochi per risolvere i quali *è necessario tornare sui propri passi*.

Così non era per la torre di Hanoi: lì, una volta capito l'algoritmo di risoluzione, non occorreva mai «rimangiarsi» una scelta fatta.

Per il Sudoku *un algoritmo del genere non esiste*. L'unico algoritmo universale per risolvere qualsiasi schema Sudoku è andare per tentativi.

La conseguenza di questo fatto è poco allegra per il risolutore: una volta inserita una cifra a matita per vedere che cosa succede andando avanti, ci si troverà ben presto di fronte a un nuovo bivio. Di nuovo bisognerà fare una supposizione, e vedere poi se andando avanti si incontra una casella impossibile da riempire. Le supposizioni così si annideranno una dentro l'altra, come i rami di un albero.

La figura 6 della prossima pagina mostra un esempio di «albero di ricerca della soluzione». In alto ci troviamo di fronte a una scelta multipla – indicata dal punto interrogativo – tra le cifre 3, 6 e 8.

Proviamo allora a inserire il 3 e andiamo avanti, finché non ci troviamo di fronte a una nuova scelta multipla, relativa a un'altra casella, tra le cifre 1 e 3.

Proviamo ancora con il 3, perché ci piace questo numero, e andiamo avanti. Alla scelta successiva – tra 1, 4 e 5 – scegliamo di inserire il 5. A quella ancora successiva – tra 1, 3, 7 e 8 – scegliamo di nuovo il 3.

Andando ancora avanti, arriviamo però a una casella in cui non si può mettere alcuna cifra senza violare le regole: è questo il significato del cartello STOP.

Che si fa a questo punto? Bisogna tornare indietro all'ultima scelta fatta e provare 1, 7 oppure 8 al posto del 3. Se tutte queste possibilità conducono ancora a vicoli ciechi, dovremo ritornare alla scelta precedente, in cui tra 1, 4 e 5 avevamo scelto 5, e cambiarla.

Se tenete conto del fatto che il Sudoku ha 81 caselle, capirete facilmente che il vostro albero di ricerca diventerà molto, molto folto.

In effetti il numero dei percorsi possibili dipende *in modo esponenziale* dal numero di caselle del Sudoku (che possono anche essere più di 81). E poiché il numero dei percorsi possibili è proporzionale al tempo di risoluzione, anche il Sudoku ha un tempo di risoluzione che cresce esponenzialmente all'aumentare della grandezza dello schema.

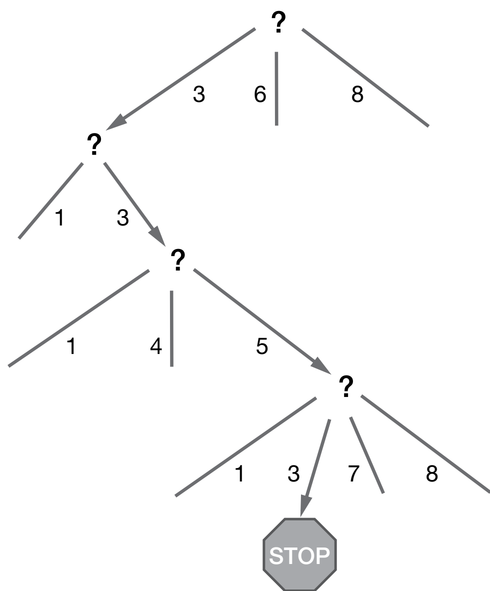

Figura 6. Un esempio di «albero di ricerca della soluzione» di un Sudoku.

Qui non ci addentreremo nel problema di che cosa voglia dire esattamente costruire uno schema Sudoku «più grande».

Possiamo però immaginare senza difficoltà un Sudoku in cui, invece di 9 quadrati di lato 3, ci siano 16 quadrati di lato 4; se ne vedono ogni tanto sulle riviste di enigmistica. In un Sudoku di questo genere, chiaramente, le cifre diverse da inserire non sono più 9, ma 16; si potrebbero usare per esempio le lettere dalla A alla P. Il procedimento si può generalizzare anche a Sudoku non quadrati, ma questo discorso ora non ci interessa.

Vogliamo invece concentrare l'attenzione sul fatto che, poiché il tempo di risoluzione dipende in modo esponenziale dal numero delle caselle, per schemi molto grandi nemmeno i nostri computer più veloci saranno in grado di trovare una soluzione in un tempo ragionevole.

Come la torre di Hanoi «con troppi dischi», anche il Sudoku «con troppe caselle» si rivela un problema *intrattabile*. E per arrivare al «troppo» è sufficiente un numero di variabili assai piccolo.

Ribadiamo un concetto che sarà molto importante nel seguito: il Sudoku è un problema la cui soluzione richiede *generalmente* un tempo molto lungo, perché esistono molte strade possibili da percorrere.

Può anche capitare però che il tempo sia brevissimo, per esempio nel caso facile in cui tutte le mosse siano obbligate; ma anche nel caso più difficile, potreste essere così fortunati da azzeccare tutte le scelte giuste al primo tentativo!

Anche senza far conto sulla fortuna, ci possono essere strategie per scegliere bene le supposizioni. È vero che qualsiasi algoritmo dovrà esplorare per tentativi l'albero delle soluzioni, ma ci sono esplorazioni più o meno intelligenti. L'abilità del solutore sta nella scelta dei tentativi da fare *per primi*: un buon algoritmo è quello che inizialmente esclude i tentativi che, per varie ragioni, appaiono meno promettenti. Torneremo più avanti sul problema della costruzione di buoni algoritmi.

C'è un'altra caratteristica che si rivelerà importante nel seguito. Sebbene per risolvere uno schema Sudoku sia necessario in generale un tempo che aumenta esponenzialmente con il numero di caselle, basta invece molto poco per verificare se una data soluzione è giusta o sbagliata: è sufficiente controllare che in ogni quadrato piccolo, in ogni riga e in ogni colonna non ci siano cifre che si ripetono.

Quest'operazione richiede un tempo breve anche per un Sudoku molto grande.

Per lo schema da 9 x 9 dobbiamo controllare per esempio 9 righe, 9 colonne e 9 quadrati, ciascuno contenente 9 cifre, facendo quindi un numero di operazioni dell'ordine di $9 \times 9 + 9 \times 9 + 9 \times 9 = 3 \times 9^2$.

Per lo schema 16 x 16 le operazioni necessarie saranno dell'ordine di 3×16^2.

Detto N il numero di caselle del lato del quadrato dello schema, avremo cioè un tempo dell'ordine di grandezza di N^2. Questo tempo *non è* esponenziale, perché N non compare nell'esponente. N^2 non è un numero che cresce spropositatamente all'aumentare di N: la verifica sarà un problema accessibile anche per N molto grande. In casi come questo si dice che il tempo cresce in modo *polinomiale*.*

Problemi di questo tipo, per i quali cioè il tempo di verifica di una soluzione è polinomiale, si chiamano – per motivi che vedremo più avanti – *problemi NP*.

È stato dimostrato anzi che il Sudoku appartiene a una classe particolare di problemi NP: è quella dei cosiddetti *problemi NP-completi* che, come vedremo tra breve, sono la bestia nera dei matematici che studiano i problemi «intrattabili».

La fattorizzazione

Alle scuole elementari si impara a riconoscere se un numero è pari o dispari, o se è divisibile per 5: basta guardare l'ultima cifra. I bambini scoprono eccitati di saper riconoscere al volo la parità anche di numeri molto grandi. Dal punto di vista computazionale, questo problema è semplicissimo: richiede *una sola* operazione, per grande che sia il numero considerato.

* I polinomi sono quelle espressioni in cui N compare come base di una potenza, con un esponente magari anche grande (N^2, N^3 o perfino N^{100}), ma mai all'esponente (come in 2^N).

Alle scuole medie poi ci insegnano a riconoscere se un numero è divisibile per 3, per 9, per 7. Anche queste sono operazioni che si possono fare senza grande difficoltà su numeri molto grandi: per esempio, per sapere se 71 170 773 è divisibile per 3, basta sommare le sue cifre. Si ottiene 33, e poiché 33 è divisibile per 3, lo è anche il nostro numero iniziale.

In questo caso abbiamo fatto un numero di operazioni praticamente uguale alle cifre del numero (7 addizioni tra le 8 cifre del numero, più la «valutazione finale» sul 33). Per un numero molto più grande, le cose non cambiano: al massimo dovremo iterare, cioè ripetere la procedura, perché la divisibilità per 3 del numero corrispondente alla somma delle cifre non è più valutabile «a occhio». Ma si può dimostrare che un computer può svolgere questa procedura in un tempo che cresce in maniera *polinomiale* al crescere delle cifre del numero iniziale.

Una domanda di ben altra portata è chiedersi se un numero sia *primo* o meno (un numero è primo quando è divisibile soltanto per 1 e per sé stesso).

I numeri primi ci affascinano da millenni: malgrado siano oggetti matematici molto semplici da definire, hanno un comportamento difficilissimo da spiegare. Appaiono in maniera irregolare nella sequenza dei numeri naturali e, sebbene si sia identificata qualche regolarità nel loro comportamento *medio*, ancora oggi non siamo in grado di prevedere quanti e quali numeri primi ci saranno tra due numeri dati.

Ma torniamo alla nostra domanda: come si riconosce se un dato numero è primo?

Prendiamo per esempio 16 847. Una prima idea può essere di applicare *rigorosamente* la definizione di numero primo: proviamo cioè a dividere il nostro numero per *tutti* i numeri minori di esso e vediamo se ce n'è qualcuno (diverso da 1 e dal numero stesso) per cui è divisibile. In questo caso sarebbero necessarie 16 845 divisioni (tutti i numeri tra 2 e 16 846).

Ma subito ci rendiamo conto che non occorre provare *tutti* i numeri, bastano i numeri primi: se un numero non è divisibile per 2, infatti, non sarà divisibile nemmeno per 4, 6, 8, e così via. E se non è divisibile per 3 non sarà divisibile nemmeno per 6, 9, 12, e così via.

Questa strategia per risparmiare tempo ci fa però imbattere in una strana circolarità: come faccio a sapere *quali sono* i numeri primi per cui dividere?

In questo caso però siamo fortunati: dopo aver ottenuto risultati decimali per le divisioni per 2, 3, 5, 7, 11, 13, ci accorgiamo che 16 847 : 17 = 991. Dunque il numero 16 847 *non* è primo.

Per scoprirlo abbiamo fatto esattamente 7 divisioni. Se non avessimo adottato la scorciatoia di dividere per i soli numeri primi, avremmo fatto 16 divisioni.

Notate che questo algoritmo non ci dice soltanto che il numero non è primo: ci fornisce anche uno dei numeri per cui è divisibile. In questo caso siamo proprio fortunati: poiché sia 17 sia 991 sono primi, abbiamo ottenuto anche la *fattorizzazione* del nostro numero iniziale, ossia i numeri primi di cui esso è il prodotto, nel nostro esempio 17 e 991. Uno dei due è piccolo, perciò l'abbiamo trovato subito.

Il caso peggiore è quando abbiamo un numero pari al prodotto di due fattori primi uguali: per esempio 10201 si ottiene dalla moltiplicazione 101 x 101. Per trovare questa fattorizzazione occorre provare a dividere 10201 per tutti i numeri primi fino a 101, cioè fino alla sua radice quadrata.

Nel caso peggiore possibile, dunque, dato un numero N dobbiamo dividerlo per tutti i numeri primi minori di $\sqrt{N}$ per scoprire se è primo o meno.

Resta aperto il problema di come facciamo a conoscere tutti i numeri primi minori di $\sqrt{N}$. Dato che in generale non li conosciamo, saremo costretti ad applicare ricorsivamente l'algoritmo finché non raggiungiamo numeri abbastanza piccoli da apparire in qualche pubblicazione (esistono vere e proprie «tavole dei numeri primi»). Se questo ci rallenta troppo, dovremo applicare l'algoritmo più «stupido» di dividere per tutti i numeri. Ne riparleremo nel prossimo capitolo.

Che cosa succede al tempo di risoluzione del problema della fattorizzazione di un numero, quando le cifre del numero aumentano?

Per rispondere osserviamo che, una volta trovato il primo fattore, il problema non è risolto ma dobbiamo continuare ad applicare l'algoritmo al secondo fattore, e così via.

Facendo una stima molto rozza, possiamo dire che per ogni due cifre che aggiungiamo al numero iniziale, esso diventa come ordine di grandezza cento volte più grande, dunque la sua radice quadrata diventa come ordine di grandezza dieci volte più grande.

Perciò il numero di operazioni necessarie per trovare il primo fattore si decuplica, all'incirca, ogni volta che aggiungiamo due cifre.

Questa, ahimé, è una crescita esponenziale: aggiungendo 20 cifre, il numero delle operazioni crescerà di un fattore 10 000 000 000, cioè dieci miliardi di volte! E questo senza considerare i fattori successivi al primo.

La fattorizzazione quindi è un problema il cui tempo di risoluzione cresce in maniera esponenziale al crescere delle cifre del numero. Anche se le cifre non sono tantissime, neppure con i più potenti computer saremo in grado di fattorizzare il numero.

Esistono in realtà diversi algoritmi più astuti per la fattorizzazione, in cui il tempo di risoluzione si riduce notevolmente. Ma nessuno di essi riesce a fattorizzare in un tempo polinomiale nel numero di cifre.

Le cose vanno meglio se si vuole sapere soltanto *se* un numero è primo: sono stati sviluppati di recente algoritmi in grado di stabilirlo in un tempo polinomiale nel numero di cifre.

Del resto, come per il Sudoku, anche per la fattorizzazione è assolutamente banale verificare se una certa soluzione sia giusta. Se vi chiedo di fattorizzare il numero 17 179 607 041, vi metterò in seria difficoltà; ma se vi chiedo soltanto di verificare che la sua fattorizzazione è 131 071 x 131 071, potrete farlo in un attimo. In effetti anche la fattorizzazione è un problema NP, cioè un problema la cui soluzione è verificabile in un tempo polinomiale.

Ovviamente esistono numeri con tantissime cifre la cui fattorizzazione è banale. Se per esempio il numero è una potenza di 2 anche molto grande, lo fattorizzeremo immediatamente dividendo sempre per 2.

I numeri difficili da fattorizzare sono quelli che hanno fattori primi molto grandi. Per motivi che spiegheremo più avanti, questi numeri possono essere molto utili; e di conseguenza i numeri primi grandi sono una merce molto richiesta. Non in senso figurato: ci sono davvero aziende che *vendono numeri primi*, e a caro prezzo!

In effetti, trovare questi numeri è davvero molto difficile. Notizie sulla «caccia» ai più grandi numeri primi si possono trovare sul sito web *The prime pages* dell'Università del Tennessee. La pagina viene aggiornata quotidianamente: nel momento in cui scrivo il più grande numero primo conosciuto ha ben 17 425 170 cifre e vale $2^{57\,885\,161}-1$.[**]

Ottimizzare

Non è raro imbattersi in problemi concreti la cui risoluzione necessita di un approccio simile a quello del Sudoku, cioè richiede di andare per tentativi con gomma e matita, tornando sui propri passi quando necessario.

** È bene specificare che, quando diciamo «il più grande numero primo conosciuto», ciò non implica che si conoscano tutti i numeri primi minori di esso.

L'esempio più classico è la preparazione dell'orario dei professori di una scuola: non soltanto il malcapitato che deve redigerlo si trova a dover soddisfare una miriade di richieste contemporanee, ma ogni volta che modifica qualche cosa, questa modifica ha un effetto a valanga su tutto il resto. Quello che abbiamo chiamato «albero delle soluzioni» è per un problema del genere ancora più complicato che per il Sudoku.

Redigere un orario scolastico è uno dei cosiddetti *problemi di ottimizzazione*, che spesso hanno un tempo di risoluzione che aumenta esponenzialmente con il numero di variabili.

L'esempio più famoso di problema di ottimizzazione è il cosiddetto *problema del commesso viaggiatore*: deve recarsi in certo numero di città, per esempio Roma, Parigi, Londra, Madrid, Berlino, Copenhagen e Lisbona. Qual è il suo percorso ideale?

In generale sarà quello che minimizza un certo «costo». Il costo potrebbe essere la distanza percorsa, il tempo impiegato o il costo totale dei biglietti. Alcune città per esempio potrebbero essere collegate tra loro da compagnie low-cost, perciò sarebbe conveniente seguire le rotte aeree, anche se così si allunga il percorso.

Prendiamo come «costo» di riferimento la distanza ferroviaria tra le città. Il nostro problema sarà dunque quello di passare per tutte e sette le città percorrendo la più breve distanza possibile, per esempio partendo da Roma e ritornando a Roma alla fine del giro.

Tabella 1. Le distanze ferroviarie in km tra alcune città europee.

	Roma	Parigi	Londra	Madrid	Berlino	Copenaghen
Parigi	1421					
Londra	1808	414				
Madrid	1969	1305	1722			
Berlino	1516	1050	1051	2358		
Copenaghen	1867	1189	1189	2518	384	
Lisbona	2592	2347	2347	626	2985	3145

Proviamo a ipotizzare un itinerario ragionevole, usando come riferimento la tabella 1.

Da Roma potremmo per esempio recarci prima a Parigi, poi a Madrid, quindi a Lisbona, Londra, Berlino e infine Copenaghen, per poi tornare a Roma. Sommando le distanze, risulta che percorreremo 9001 kilometri.

Si può fare di meglio? Per tentativi, ci accorgiamo di sì: passando prima per Lisbona e poi da Madrid, Londra, Parigi, Berlino e Copenaghen, percorreremo «soltanto» 8655 kilometri.

Come facciamo a stabilire se questo percorso è il più breve possibile? Ebbene, a quanto pare non c'è altro modo che confrontarlo con tutti gli altri percorsi possibili. Che però non sono pochi: con le nostre sette città abbiamo 6 x 5 x 4 x 3 x 2 percorsi possibili, cioè 720 percorsi in tutto. Stabilita la città di partenza, infatti, ci sono 6 scelte possibili per la seconda città, poi restano 5 scelte possibili per la terza città, e così via.

In matematica il prodotto di un numero intero per tutti gli interi inferiori è chiamato *fattoriale* e si indica con un punto esclamativo. Così, per esempio, 6 x 5 x 4 x 3 x 2 x 1 = 6! e si legge «sei fattoriale».

In generale, nel problema del commesso viaggiatore per *N* città avremo (*N*–1)! percorsi possibili. Ma la funzione fattoriale cresce ancora più velocemente della funzione esponenziale, al crescere di *N*: dunque, all'aumentare del numero di città, dovremo confrontare un numero di percorsi che aumenta a dismisura.

Al crescere del numero delle città, il tempo necessario per confrontare i percorsi diventa presto insostenibile.

Bastano 15 città per avere ben 87 178 291 200 percorsi da confrontare; e con sole 50 città il numero di percorsi diventa dell'ordine di 10^{63}, troppo grande anche per i più potenti computer di oggi.

Il problema del commesso viaggiatore è anch'esso, come il Sudoku, un problema NP-completo (spiegheremo fra breve il significato di quest'espressione). Come per il Sudoku, anche in questo caso siamo costretti a procedere per tentativi. Anzi, qui è molto peggio: dato che vogliamo trovare *la migliore* soluzione, saremo sempre costretti a esaminare *tutte* le possibilità.

Anche per il problema del commesso viaggiatore sono stati inventati algoritmi che consentono di «scegliere bene» fra i percorsi possibili, evitando per esempio di sondare percorsi palesemente troppo lunghi, risparmiando così un po' di tempo di calcolo.

Ma anche per l'algoritmo più efficiente finora inventato il tempo di esecuzione cresce comunque in maniera esponenziale al crescere del numero di città.

Per inciso, per le sette città della tabella 1 basta un programmino da personal computer per analizzare i 720 percorsi (7 è ancora un numero di città abbastanza piccolo): il percorso migliore risulta essere Roma, Lisbona, Madrid, Parigi, Londra, Copenhagen, Berlino e di nuovo Roma, per un totale di 8026 km. Invertendo Madrid e Lisbona i kilometri diventano 8030, appena 4 in più. Se decidiamo di non preoccuparci di differenze così piccole, potremo risparmiare moltissimo tempo di calcolo accontentandoci di algoritmi che trovano soluzioni «buone», anche se non ottimali.

Il problema del commesso viaggiatore descritto finora non è molto realistico; ma basta usare una funzione «costo» appena più verosimile – tenendo conto per esempio di variabili come tariffe e orari – per avere un problema che si presenta davvero nella vita reale.

Le compagnie aeree, per esempio, lo affrontano quotidianamente quando devono decidere come spostare da una città all'altra i velivoli e il personale: ogni aereo che viaggia vuoto e ogni hostess che viaggia come passeggero è una perdita economica.

Lo stesso problema si presenta ai progettisti elettronici, quando devono disporre i componenti all'interno di un circuito cercando la disposizione ottimale in termini di spazio occupato e materiale necessario. Spesso in questi casi ci si deve accontentare di una soluzione «buona», giacché come detto quella ottimale è impossibile da trovare.

Come si intuisce da questi esempi, la teoria matematica della complessità computazionale è un problema molto attuale e rilevante, con ricadute economiche di grande portata.

Dal punto di vista teorico, quando si parla di complessità computazionale ci si riferisce al caso *peggiore* del problema in esame. Nella realtà non è detto che ci si trovi in questo caso. Se per esempio le città da visitare fossero tutte allineate – come Piacenza, Parma, Modena e Bologna – il problema diventerebbe banale. E se le città fossero tre in Europa e tre in America, è chiaro che il percorso da fare dovrebbe evitare la ripetizione di traversate oceaniche, quindi non occorrerebbe esaminare davvero tutte le possibilità: potremmo limitarci a quelle che evitano più di una traversata.

Così nella realtà questi problemi «difficili» si affrontano valutando caso per caso la strategia migliore da applicare. Ma disporre di una soluzione teorica al caso peggiore del problema generale sarebbe tutta un'altra cosa.

Le classi di difficoltà computazionale

Cerchiamo di fare il punto sui problemi in cui ci siamo imbattuti finora. Abbiamo incontrato problemi il cui tempo di risoluzione aumenta in modo polinomiale con il numero delle variabili: è il caso della divisibilità per 3 di un numero. Questi problemi appartengono

a una classe di difficoltà chiamata P (da *polinomiale*) e, sebbene non siano necessariamente facili da risolvere con carta e penna, non comportano particolari problemi per un computer, a meno che il numero di variabili in gioco diventi davvero enorme.

Abbiamo poi incontrato problemi il cui tempo di risoluzione aumenta in modo esponenziale con il crescere delle variabili, e abbiamo visto che per questi problemi il tempo di risoluzione diventa enorme anche per un numero di variabili relativamente piccolo; tuttavia basta un tempo polinomiale per verificare se una soluzione sia giusta. Questi problemi appartengono a una classe di difficoltà chiamata NP, da *nondeterministic polynomial*.

Per capire l'origine della sigla NP, ritorniamo per un attimo al Sudoku. Immaginate di non essere soli a risolvere il rompicapo, ma di avere un esercito illimitato di collaboratori. Non appena vi trovate di fronte a un bivio tra due (o più) opzioni per una casella, convocate due (o più) collaboratori e li istruite a procedere ciascuno con un tentativo diverso. Naturalmente, ogniqualvolta uno dei due (o più) si imbatterà in un nuovo bivio, dovrà convocare altri collaboratori, e così via. Ebbene: prima o poi *qualcuno* risolverà il Sudoku, senza che nessuno abbia mai dovuto fare una sola cancellatura!

Così, se si ha a disposizione un numero adeguato di collaboratori, il tempo di risoluzione del Sudoku diventa polinomiale. La controindicazione è che ora aumenta in maniera esponenziale il numero dei collaboratori necessari…

Di fatto, si tratta di lavorare *in parallelo* e non *in serie*. Questa modalità di lavoro richiede più persone, ma permette di fare molto più in fretta. Gli algoritmi basati sul lavoro in parallelo sono detti «non deterministici». Ecco dunque che cosa significa *nondeterministic polynomial*: un problema è NP se è risolvibile in maniera polinomiale con un algoritmo non deterministico.

I nostri computer sono costruiti in modo da lavorare *in serie*: non possono fare cioè più operazioni contemporaneamente. Con il vostro pc potete tenere aperti insieme più programmi, ma ogni processore lavora *su uno solo alla volta*. Ecco perché con molte applicazioni aperte il computer «rallenta»: può dedicare soltanto poco tempo a ognuna.

Invece i cosiddetti *computer quantistici* sono in grado di lavorare – in un certo senso – in parallelo. Con questo tipo di computer, perciò, i problemi NP diventano veloci da risolvere. Sfortunatamente i computer quantistici non esistono. La teoria della computazione quantistica progredisce di anno in anno, ma pare che si sia ancora molto lontani dall'arrivare effettivamente a costruire uno di questi computer.[***]

Abbiamo dunque inserito i nostri problemi in due classi: P e NP. Attenzione, però, non si tratta di

*** Curiosamente, proliferano nel frattempo gli algoritmi per computer quantistici: sebbene questi non esistano ancora, la gente sta già ragionando su come utilizzarli. Fin dal 1994 per esempio è stato inventato un algoritmo per risolvere con un computer quantistico il problema della fattorizzazione.

classi separate: anzi, la prima è addirittura contenuta nella seconda! Infatti, se un problema è risolvibile in tempo polinomiale con un algoritmo «in serie» (deterministico), lo sarà a maggior ragione con un algoritmo «in parallelo» (non deterministico).

In altre parole: se possiamo verificare in tempo polinomiale le soluzioni di problemi NP come il Sudoku o la fattorizzazione, saremo a maggior ragione in grado di verificare in tempo polinomiale le soluzioni di problemi come la divisibilità per 3.

Per tutti questi problemi le soluzioni si possono *verificare* in un tempo polinomiale, che è una caratteristica della classe NP, ma soltanto per alcuni (quelli che appartengono alla classe P) le soluzioni si possono anche *trovare* in un tempo polinomiale. P è dunque un sottoinsieme di NP, come mostra la figura 7.

Tra i problemi NP abbiamo già accennato alla particolare sottoclasse dei problemi *NP-completi*.

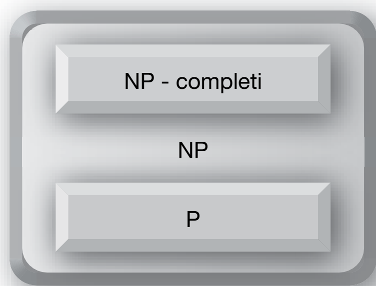

Figura 7. Le classi di difficoltà computazionale.

Senza addentrarci nei dettagli di questa distinzione, diremo soltanto che in questa classe ci sono i problemi NP più difficili in assoluto. E non è tutto: questi problemi, in apparenza molto diversi tra loro (pensate a quanto sembrano diversi il Sudoku e il problema del commesso viaggiatore), sono in realtà matematicamente tutti riconducibili l'uno all'altro.

Ciò significa che, se si trovasse anche *un solo algoritmo* in grado di risolvere in un tempo polinomiale *un solo problema* NP-completo, per esempio il problema del commesso viaggiatore, allora tutti gli altri problemi NP-completi sarebbero risolvibili in un tempo polinomiale.

Ma c'è di più: qualsiasi altro problema NP può essere ricondotto a un problema NP-completo. Risolti in tempo polinomiale gli NP-completi, dunque, sarebbero risolti in tempo polinomiale *tutti* i problemi della classe NP. Cioè, in conclusione, la classe NP sarebbe uguale a P.

Come aggiudicarsi un milione di dollari

Finora abbiamo dato per scontato che non esistano algoritmi capaci di risolvere in tempo polinomiale un problema NP. Ma in realtà ciò *non è mai stato dimostrato*. Potrebbe essere che non siamo stati abbastanza bravi? Potrebbe la creatività umana inventare un modo geniale di risolvere il peggior Sudoku, o il caso più generale del problema del commesso viaggiatore, in tempo polinomiale?

Non ci sono prove che ciò non possa accadere: in linea teorica, potrebbe essere davvero che P = NP e che la loro attuale distinzione derivi solo dalla nostra incapacità di trovare l'algoritmo «giusto». Detto questo, la maggior parte degli scienziati è fermamente convinta che P e NP siano due classi diverse. Ma, si sa, in matematica non basta essere convinti: bisogna fornire prove.

Sul sito del Clay Mathematics Institute trovate l'elenco dei *Millennium Problems*: sono sette problemi, per ciascuno dei quali l'istituto ha messo in palio un milione di dollari. Uno di questi problemi è quello del rapporto tra le classi P e NP: il premio andrà a chi riuscirà a dimostrare o che sono diverse, oppure che sono uguali.

Notate che per dimostrare l'uguaglianza basterebbe inventare *un singolo* algoritmo in grado di risolvere in tempo polinomiale *un singolo* qualsiasi problema NP-completo. La dimostrazione della diversità richiede probabilmente un procedimento molto più complesso: ma lo sapremo soltanto se e quando tale dimostrazione sarà stata trovata.

Per la cronaca, uno dei sette problemi – la cosiddetta «congettura di Poincaré» – è stato risolto nel 2003 dal matematico russo Grigorij Perel'man, il quale ha poi peraltro rifiutato sia il milione di dollari, sia l'ambitissima Medaglia Fields che gli era stata attribuita. Gli altri sei *Millennium Problems* sono tuttora irrisolti.