PIANO LAUREE SCIENTIFICHE

LABORATORIO DI COMPLESSITÀ COMPUTAZIONALE

LEZIONE 6

GIORGIO GAMBOSI
DIPARTIMENTO DI MATEMATICA

UNIVERSITÀ DI ROMA "TOR VERGATA"

1. Risoluzione e verifica di problemi

La classificazione dei problemi decidibili ha in realtà una zona grigia localizzata tra i problemi trattabili e quelli intrattabili. Esistono decine di migliaia di esempi interessanti di problemi che giacciono in tale zona grigia: di questi ne riportiamo uno tratto dal campo dei solitari e relativo al noto gioco del *Sudoku*.

In tale solitario, il giocatore è posto di fronte a una tabella di nove righe e nove colonne parzialmente riempita con numeri compresi tra 1 e 9: la tabella è suddivisa in nove sotto-tabelle, ciascuna di tre righe e tre colonne. Il compito del giocatore è quello di riempire le caselle vuote della tabella con numeri compresi tra 1 e 9, rispettando i seguenti vincoli:

(1) ogni riga contiene tutti i numeri compresi tra 1 e 9;
(2) ogni colonna contiene tutti i numeri compresi tra 1 e 9;
(3) ogni sotto-tabella contiene tutti i numeri compresi tra 1 e 9.

Nella soluzione di questo problema, il giocatore è tipicamente costretto a eseguire un algoritmo di **backtrack**, in base al quale effettua una sequenza di scelte, ognuna delle quali corrisponde ad inserire un numero in una casella ancora vuota, senza violare i vincoli posti

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

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

Figura 1 Un esempio di istanza del gioco del Sudoku e la corrispondente soluzione.

dal problema. La scelta operata più recentemente (se non conduce a una soluzione del problema) viene annullata e sostituita con un'altra scelta possibile (che non sia già stata analizzata). Questo, nel caso peggiore, equivale a considerare tutte le possibili assegnazioni di numeri alle caselle vuote (che supponiamo siano m) e a verificare, per ogni assegnazione considerata, se i vincoli precedenti sono rispettati.

```
FOR ALL assegnazioni di numeri alle m caselle vuote {
  IF (l'assegnazione attuale rispetta i vincoli) {
    RETURN l'assegnazione
  }
}
```

Una assegnazione di numeri alle caselle vuote può essere espressa come una sequenza di m caratteri, ognuno nell'insieme $\{1, \ldots, 9\}$.

Quante sono le possibili assegnazioni?

Soluzione: Ogni assegnazioni ha 9 possibili valori per il primo carattere, 9 per il secondo, e così via, fino a 9 per l'm-esimo: ciò ci dà un totale di 9^m possibili assegnazioni. Per ogni assegnazione, l'algoritmo deve verificare il rispetto dei vincoli.

Quante caselle bisogna osservare per verificare i vincoli?

Soluzione: Le caselle da osservare sono 9 per ognuno dei vincoli, per un totale di $3 \times 9 = 27$. L'algoritmo esegue quindi, nel caso considerato, 27×9^m passi (osservazioni di caselle). In generale, usando n cifre (invece delle 9 considerate finora), il gioco necessita di una tabella di dimensione $n \times n$: in questo caso l'algoritmo esegue $3n \cdot n^m$ osservazioni. Dato che necessariamente $m \leq n^2$, possiamo anche dire che al più vengono osservate $3n \cdot n^{n^2} = 3n \cdot 2^{\log_2 n^{n^2}} = 3n \cdot 2^{n^2 \log_2 n}$ caselle, e quindi l'algoritmo suddetto esegue una quantità di passi (più che) esponenziale rispetto alla dimensione della tabella.

A differenza del problema delle Torri di Hanoi, e similarmente al problema della disposizione dei rissosi cavalieri della tavola rotonda, non possiamo però concludere che il problema del gioco del Sudoku sia intrattabile: nonostante si conoscano solo algoritmi esponenziali per il Sudoku, nessuno finora è riuscito a dimostrare che tale problema possa ammettere o meno una risoluzione mediante algoritmi polinomiali.

Una caratteristica importante del Sudoku, pur nella difficoltà di risoluzione, è data dalla semplicità di verifica di una soluzione.

Supponiamo infatti che, stanchi di tentare di riempire una tabella di dimensione $n \times n$ pubblicata su una rivista di enigmistica, incominciamo a nutrire dei seri dubbi sul fatto che tale tabella ammetta una soluzione. Per tale motivo, decidiamo di rivolgerci direttamente all'editore chiedendogli di convincerci che è effettivamente possibile riempire la tabella. Ebbene, l'editore ha un modo molto semplice di fare ciò: ci invia la sequenza delle cifre da inserire nelle caselle vuote. Tale sequenza ha chiaramente lunghezza $m \leq n^2$ ed è quindi polinomiale rispetto a n: inoltre, possiamo facilmente verificare la correttezza del problema

proposto dall'editore, riempiendo le caselle vuote con le cifre della sequenza ed effettuando la relativa verifica.

Quanti passi bisogna effettuare per la verifica, data la soluzione?

Soluzione: Evidentemente, per la verifica sono necessari m passi per riempire le caselle vuote con i valori forniti dalla soluzione, seguiti da $3n$ osservazioni per controllare il rispetto dei vincoli su righe, colone e sotto-tabelle: ciò fornisce un totale di $m + 3n \leq n^2 + 3n$ passi. L'algoritmo di verifica è quindi polinomiale.

In definitiva, *verificare* che una sequenza di m cifre sia una soluzione di un'istanza del Sudoku può essere fatto in tempo polinomiale mentre, ad oggi, nessuno conosce un algoritmo polinomiale per *trovare* una tale sequenza.

La stessa caratteristica è presentata dal problema della disposizione dei rissosi cavalieri della tavola rotonda. Anche in questo caso non è noto nessun algoritmo polinomiale per trovare una disposizione (se esiste): ricordiamo che l'algoritmo che abbiamo considerato richiedeva di effettuare $n!$ verifiche. Al tempo stesso, supponiamo che il potente mago Merlino comunichi ad Artù di avere trovato, per magia, una disposizione: Artù può allora verificare se la disposizione trovata da Merlino è effettivamente una soluzione del problema eseguendo ...

Quanti passi deve effettuare Artù per verificare la disposizione suggerita da Merlino?

Soluzione: Artù deve semplicemente controllare tutte le coppie di cavalieri che si trovano vicini nella disposizione di Merlino, per un totale di n verifiche, e quindi in un numero polinomiale di passi.

Questa caratteristica di non avere algoritmi efficienti di risoluzione, unita alla semplicità di verifica di possibili soluzioni è comune a decine di migliaia di problemi che ricorrono in situazioni reali, che vanno dall'organizzazione del trasporto a problemi di allocazione ottima di risorse. Presentano questa caratteristica, ad esempio, i problemi seguenti:

1.1. **L'oculato commesso viaggiatore.** Dato un insieme di città e specificato, per ogni coppia di città, la distanza chilometrica per andare dall'una all'altra o viceversa, un commesso viaggiatore si chiede se sista un modo per visitare tutte le città una e una sola volta, tornando al termine del giro alla città di partenza e percorrendo al più un certo numero di chilometri. Consideriamo, ad esempio, la seguente istanza del problema in cui sono considerate 9 città olandesi:

	A	B	D	E	H	L	M	R	U
Amsterdam	0	101	98	121	20	55	213	73	37
Breda		0	30	57	121	72	146	51	73
Dordrecht			0	92	94	45	181	24	61
Eindhoven				0	136	134	86	113	88
Haarlem					0	51	228	70	54
L'Aia						0	223	21	62
Maastricht							0	202	180
Rotterdam								0	57
Utrecht									0

E assumiamo che il commesso viaggiatore voglia provare a visitare tutte le città percorrendo non più di 600 chilometri: se decide di percorrere le città secondo il loro ordine alfabetico, allora percorre un numero di chilometri pari a

$$101 + 30 + 92 + 136 + 51 + 223 + 202 + 57 + 37 = 929$$

Supponiamo, invece, che decida di percorrerle nel seguente ordine: Amsterdam, Haarlem, L'Aia, Rotterdam, Dordrecht, Breda, Maastricht, Eindhoven e Utrecht. In tal caso, il commesso viaggiatore percorre un numero di chilometri pari a

$$20 + 51 + 21 + 24 + 30 + 146 + 86 + 88 + 37 = 503$$

Tra l'altro, considerando tutte le possibili $9! = 362880$ permutazioni delle nove città, potremmo anche verificare che quest'ultima è la soluzione migliore possibile. Sfortunatamente, il commesso viaggiatore non ha altra scelta che applicare un algoritmo esaustivo (cioé sostanzialmente provare tutte le possibilità) per trovare la soluzione al suo problema: ciò ci fornisce, per il caso generico di n città e distanza massima d, un algoritmo che effettua su ognuno degli $n!$ possibili percorsi una verifica, in n passi, che la corrispondente lunghezza è minore o uguale di d. Ne risulta un numero complessivo di $n \cdot n!$ passi.

Al tempo stesso, dato un possibile percorso, il commesso viaggiatore può facilmente verificare, in tempo n, se il percorso in questione è effettivamente più breve di d.

1.2. **Ancora le torri di Hanoi.** Se consideriamo il problema delle torri di Hanoi, inteso come determinare se, dati 3 pioli e n dischi, è possibile spostare tutti i dischi dal piolo 1 al piolo 3 senza sovrapporre dischi più grandi a dischi più piccoli in al più K mosse, allora possiamo vedere che una soluzione è una sequenza di K mosse, che è verificabile mediante K passi e quindi in tempo non polinomiale rispetto alla descrizione del problema.

Quanto è lunga la descrizione di una istanza del problema delle torri di Hanoi?

Soluzione: Si richiede di specificare n e K, per cui sono necessari $\log_2 n + \log_2 K$ bit, o in generale $\log_c n + \log_c K$ se abbiamo a disposizione c caratteri diversi.

1.3. **La classe NP di problemi.** L'insieme di tutti i problemi che condividono la caratteristica di avere una verificabilità efficiente (polinomiale) di possibili soluzioni prende il nome di *classe NP*: per ogni problema in NP, riassumendo, chi ha la soluzione per un'istanza del problema, può convincerci di ciò fornendo un'opportuna informazione (detta *certificato*) che ci permette di verificare, in tempo polinomiale, l'esistenza di una qualche soluzione. Si noti come, per i problemi che abbiamo considerato, il certificato corrisponde alla soluzione. Non sempre è cosi, però: ad esempio il problema di determinare se un intero n è un primo appartiene alla classe NP, ma il certificato (che esiste) non è così immediato.

Un'altra caratteristica conseguente dei problemi in NP è che chi non ha la soluzione, può comunque procedere per tentativi in tempo esponenziale, provando a generare (più o meno esplicitamente) tutti i certificati possibili.

La classe NP include (non sappiamo se in senso stretto o meno) l'insieme dei problemi trattabili, risolubili cioè in tempo polinomiale, indicato come *classe P*.

Perché P è in NP?

Soluzione: Infatti, per ogni problema che ammette un algoritmo polinomiale, possiamo usare tale algoritmo per produrre una soluzione e, quindi, un certificato polinomiale.

Il problema del Sudoku, come anche gli altri citati sopra eccetto le torri di Hanoi (che non sono in NP) e la verifica se un intero è primo, in realtà appartengono ad un insieme più ristretto detto classe dei *problemi NP-completi* (NPC). Tali problemi sono i più difficili da risolvere algoritmicamente tra i problemi della classe NP, nel senso che se scopriamo un algoritmo polinomiale per un qualsiasi problema NP-completo, allora tutti i problemi in NP sono risolubili in tempo polinomiale (ovvero la classe NP coincide con la classe P). Se invece dimostriamo che uno dei problemi NP-completi è intrattabile (e quindi che la classe NP è diversa dalla classe P), allora risultano intrattabili tutti i problemi NP-completi.

I problemi in NP (e quindi quelli NP-completi) influenzano la vita quotidiana più di quanto possa sembrare: come detto, se qualcuno mostrasse che i problemi NP-completi ammettono algoritmi polinomiali, ovvero che $P = NP$, allora ci sarebbero conseguenze in molte applicazioni di uso comune. Per esempio, diventerebbe possibile indovinare in tempo polinomiale una parola chiave di n simboli scelti in modo casuale, per cui diversi metodi di autenticazione degli utenti basati su parole d'ordine e di crittografia basata su chiave pubblica non sarebbero più sicuri (come il protocollo *SSL* adoperato dalle banche e dal commercio elettronico per le connessioni sicure nel Web).