1

Capitolo C47
complessità computazionale

Contenuti delle sezioni

a. analisi dei processi di calcolo p. 2
b. atteggiamenti nella analisi della complessità p. 5
c. tipi di analisi della complessità [1] p. 7
d. tipi di analisi della complessità [2] p. 12
e. problemi polinomiali e problemi intrattabili p. 17
f. problemi NP p. 19
g. problemi NP-completi p. 22
h. convivere con i problemi intrattabili p. 25

25 pagine

C47 0.01 La crescita della diffusione e delle potenzialità degli algoritmi ha fatto sviluppare una vasta gamma di studi con l'obiettivo di valutare i costi e i meriti di questi strumenti che oggi tanto infuicono sulla l'intera società e di cercare di stabilire una loro complessiva clssificazione in relazione alle sopra accennate valutazioni.

C47 a. analisi dei processi di calcolo

C47 a.01 La attuale vistosa e crescente mole delle attività di calcolo automatico rende di primaria importanza conoscere l'efficienza dei procedimenti di calcolo e degli algoritmi.

A questa esigenza rispondono gli studi sulla complessità delle computazioni e sulla complessità degli algoritmi.
Si tratta di studi relativamente recenti. Lo studio degli algoritmi è antichissimo e si confonde con interi capitoli della matematica; nel passato però si studiava soltanto quella che talora viene chiamata sintesi degli algoritmi, cioè la individuazione di procedimenti che consentono la soluzione di particolari problemi di calcolo; inoltre l'attenzione era rivolta quasi esclusivamente a problemi di natura numerica e geometrica.

Intorno al 1950, con l'inizio dell'utilizzo degli elaboratori elettronici, si sono sviluppate le prime **analisi di algoritmi**, cioè i primi studi sulla efficienza dei singoli algoritmi e sul confronto tra algoritmi in grado di risolvere uno stesso problema.

C47 a.02 Intorno al 1960, quando si è consolidata la percezione delle ampie possibilità applicative degli elaboratori e sono stati decisi i primi investimenti strategici nella loro promettente industria, si è iniziato uno studio sempre più sistematico e approfondito degli algoritmi.

I primi obiettivi sono stati la padronanza di intere classi di algoritmi, la individuazione di procedimenti più efficienti di quelli, in genere intuitivi, adottati nel passato e la ricerca di indicazioni sufficientemente indipendenti dal tipo di apparecchiatura usata per le elaborazioni (in quel periodo non si era ancora imposto il complesso di standard hardware e software che intorno al 1975 ha reso facile adattare un algoritmo ad un elaboratore diverso da quello della sua prima messa a punto).

Da allora, in relazione alle crescenti esigenze del mondo delle elaborazioni automatiche, lo studio degli algoritmi ha continuato a svilupparsi alla scoperta di procedimenti il più possibile efficienti o almeno comunque più vantaggiosi dei più intuitivi per la soluzione di problemi tradizionali o posti dal nuovo scenario delle attività computazionali.

In particolare sono cresciute vistosamente le indagini sulla analisi numerica, sulla geometria computazionale e sulle manipolazioni delle strutture discrete.

C47 a.03 Le ricerche sull'analisi degli algoritmi si sono intensificate a partire dagli anni '70, quando sono stati ottenuti alcuni importanti risultati che hanno consentito di definire linee di indagine più stringenti. Attualmente, nonostanti certe intrinseche difficoltà, si possiede una notevole mole di conoscenze sugli algoritmi e sulla loro complessità e questo permette di affrontare con consapevolezza una gamma di problematiche di calcolo vistosamente più ampia.

Estese panoramiche di questi risultatisono presentate in *Johnson 1990*, *Shmoys, Tardos 1995* e *Burgisser, Clausen, Shokrollahi 1997* nelle prime edizioni di *The art of computer programming* i Donald Knuth.

C47 a.04 In termini un po' generici possiamo dire che lo studio della **complessità computazionale** ha come oggetto l'impegno richiesto per eseguire le operazioni che consentono di risolvere problemi computazionali.

Data la varietà dei problemi che portano ad attività di calcolo, delle loro motivazioni e degli strumenti utilizzabili, per giungere a risultati significativi risultano necessarie molte precisazioni (che spesso riguardano scelte di compromesso).

Innanzi tutto si sente la necessità di specificare la macchina reale o ideale (ma comunque realizzabile) alla quale si affida l'esecuzione dei calcoli.
Data l'eterogeneità delle macchine utilizzabili (dai modelli astratti più ridotti all'essenziale come le macchine di Turing, ai vari modelli di elaboratori che vengono via via resi disponibili dall'industria) si può essere indotti a giudicare impossibile un confronto stringente di strumenti tanto diversi.

Occorre poi precisare cosa si deve intendere per impegno richiesto da una computazione.
A questo proposito si presentano tre alternative complementari, non riducibili l'una all'altra.
La prima riguarda il tempo di calcolo impiegato dallo strumento usato: a questo proposito si parla di misura della **complessità temporale**.
Si può poi considerare l'estensione della memoria necessaria per registrare le informazioni trattate durante i calcoli e in questo senso si parla di misura della **complessità spaziale**.

Infine è interessante esaminare l'onere richiesto per definire con precisione la procedura esecutiva, e a questo proposito si parla di valutazione della **complessità strutturale**.

C47 a.05 Per dare indicazioni sulle misure di complessità risulta anche necessario precisare a quale gamma delle istanze del problema trattato si fa riferimento.

Dato che i problemi di maggiore interesse hanno una gamma illimitata o comunque molto ampia di istanze, sarebbe opportuno fare riferimento ad una loro distribuzione.

Ancora si presentano diverse scelte alternative che andrebbero riferite alle diverse esigenze applicative e alle diverse strumentazioni disponibili.

Quando si deve eseguire un certo tipo di elaborazione su varie istanze, interessano maggiormente valutazioni medie riferite a una certa distribuzione. Spesso invece la maggiore preoccupazione è quella di non incorrere in una elaborazione eccessivamente onerosa che potrebbe mettere in crisi il contesto applicativo nel quale essa si colloca: in tal caso interessano soprattutto le situazioni più dispendiose, i **casi peggiori**.

Quando si avvia una attività di calcolo sistematica interessa capire come cresce l'onere di una elaborazione quando aumenta, prevedibilmente, l'estensione delle istanze del problema.
Emerge allora l'importanza delle **valutazioni asintotiche della complessità**, le quali aiutano a stabilire fino a che punto risulta ragionevole, ovvero non eccessivamente costoso, affrontare il tipo di calcolo in questione.

C47 a.06 Come emerge dalle precedenti considerazioni, l'analisi della complessità computazionale si presenta come una tematica vasta e con grande varietà di aspetti (oltre che fortemente motivata).

Si potrebbe anche temere che lo studio della complessità riesca a dare solo prescrizioni frammentarie, valide solo in circostanze molto particolari.

In effetti il programma dello studio della complessità computazionale è molto ambizioso e presenta una certa tendenza a svilupparsi in direzioni diverse, talora divergenti.

In realtà, come in piccola parte vedremo nel seguito, si sono ottenuti vari risultati importanti e di ampia portata, per due ordini di motivi.

Innanzi tutto vi sono risultati teorici e considerazioni sulla efficienza e versatilità attuale e progressivamente prevedibile degli elaboratori elettronici (e oggi anche delle estese reti di processori) che rendono lecito trascurare molti dettagli delle misure di complessità.

In particolare, forse sorprendentemente, è lecito non preoccuparsi eccessivamente delle differenze tra i diversi modelli delle macchine alle quali si affidano le elaborazioni.

Risulta quindi possibile puntare a valutazioni di tipo approssimato, spesso di natura asintotica, tese a individuare aspetti sostanziali di vaste classi di algoritmi.

Si rende allora necessario porsi in grado di trascurare molti elementi non essenziali dei procedimenti di calcolo e di porsi a un adeguato livello di astrazione e generalità.

Gli altri motivi che hanno consentito il successo degli studi sulla complessità stanno nella possibilità di collegare in modo soddisfacewnte situazioni pratiche circoscritte per le quali si vogliono indicazioni operative e schemi algoritmici analizzabili efficacemente.

In particolare inoltre molti risultati consentono di evitare di pianificare attività di calcolo innovative che possono incorrere in inefficienze onerose o addirittura inaccettabili.

C47 a.07 Il collegamento tra matematica discreta e studio della complessità è bidirezionale.

La matematica discreta consente di chiarire le caratteristiche strutturali delle configurazioni discrete sulle quali si fanno agire gli algoritmi, configurazioni individuabili anche nei procedimenti risolutivi di problemi riguardanti valori reali, funzioni continue e processi dinamici.

Viceversa gli studi sulla complessità portano a parametri di efficienza operativa per le famiglie di configurazioni discrete i quali sono ormai comunemente considerati indispensabili per la piena comprensione di ciascuna problematica.

C47 b. atteggiamenti nella analisi della complessità

C47 b.01 Vediamo ora con qualche dettaglio quali distinzioni si possono fare tra gli strumenti per l'esecuzione dei calcoli, quali tipi di impegno sono da valutare e quali casistiche sono da prendere in esame.

Sopra gli strumenti con i quali si possono eseguire le elaborazioni che conducono alle soluzioni si possono assumere due atteggiamenti contrapposti e divergenti.

Il primo, decisamente empirico, si occupa di quanto specifiche implementazioni di algoritmi impegnino determinate configurazioni di macchine.
Si tratta di un atteggiamento estremamente concreto il quale però, porta a conclusioni utilizzabili in una buona gamma di situazioni solo quando si sanno scegliere implementazioni e configurazioni molto significative.

In questa direzione vengono sviluppate precise tecniche per la valutazione di situazioni tipiche (**benchmarks**) al fine di ottenere raffronti significativi di processori e di sistemi concretamente funzionanti in relazione alle esigenze di ampi settori applicativi.

Il punto di vista opposto considera un modello ideale di strumento elaboratore, tipicamente una macchina di Turing; tali modelli hanno portata del tutto generale, ma presentano caratteristiche profondamente diverse da quelle delle macchine reali o che sarebbe sensato realizzare.

Questo atteggiamento per portare a conclusioni utili deve riuscire a cogliere gli aspetti che maggiormente influiscono sopra la onerosità dei procedimenti risolutivi di determinate categorie di problemi; a questo proposito si parla di **difficoltà intrinseca** di un procedimento computazionale.

Si ha anche la possibilità di assumere atteggiamenti "intermedi" che riescono a fornire indicazioni generali per insiemi di situazioni concrete piuttosto ampie e di elevato interesse.

Per questo si può fare riferimento a un elaboratore ipotetico che denoteremo con *EI*, che rispetto ai modelli effettivamente in uso presenta differenze che, anche se non possono essere trascurate, possono essere tenute sotto controllo dal punto di vista della valutazione delle complessità.

C47 b.02 L'elaboratore *EI* è individuato dalle seguenti caratteristiche:

[EI1] Le operazioni di I/O, ingresso e uscita, non gli costano nulla. Questa ipotesi è accettabile quando si studiano elaborazioni nelle quale i calcoli prevalgono sulle operazioni di I/O, ovvero quando si studiano processi che partono da dati in memoria centrale e in memoria lasciano i risultati. Questa ipotesi non ha invece senso quando si studiano elaborazioni nelle quali i trasferimenti di informazioni sono rilevanti.

[EI2] La memoria centrale di *EI* ha capacità illimitata. Questa ipotesi attualmente e in prospettiva, stante la disponibilità a costi molto contenuti di grandi quantità di memoria, per molte problematiche può essere accettata tranquillamente.

[EI3] Le celle di memoria indirizzabili consentono di registrare senza differenze operative i diversi tipi di dati elementari da elaborare (valori logici, caratteri, interi non eccessivamente grandi, reali con un numero di cifre significative non troppo alto, stringhe di lunghezze contenute).

Questa ipotesi oggi è inaccettabile solo quando si trattano interi molto grandi (superiori ai due miliardi), e quando si devono affrontare questioni critiche per la precisione, cioè quando è necessario operare con più di otto o sedici cifre decimali).

[EI4] Il tempo di accesso per ogni registro di memoria è indipendente dalla sua estensione (byte, word, semiword, doubleword, ...) e dall'ampiezza della memoria complessivamente impegnata. Questa

ipotesi è senz'altro accettabile quando si usano processori con parallelismo a 32 bits o a 64 bits (oggi prevalenti); ponevano qualche problema solo quando si usavano processori a 16 bits per elaborazioni su grandi quantità di dati.

[EI5] L'elaboratore è in grado di eseguire un buon repertorio di istruzioni (operazioni aritmetiche, confronti, salti, diramazioni, ...). Con i processori attuali anche questa ipotesi è ampiamente accettabile, in linea di massima anche per i microprocessori incorporati nelle apparecchiature di controllo e di monitoraggio.

C47 b.03 In genere si ipotizza anche che i programmi da valutare siano redatti in modo lineare, cioè non presentino grossolanità che portano inefficienze evitabili e non facciano uso di dispositivi molto particolari (coprocessori o circuiti integrati) che consentano scorciatoie inaspettate.

Si osservi che le ipotesi enunciate tendono a sgombrare il campo dai dettagli dell'hardware e dell'implementazione.

Per quanto riguarda i tempi di esecuzione dei calcoli, spesso si richiede che i diversi tipi di operazioni elementari sui diversi possibili tipi di operandi abbiano una stessa durata. In molti casi però risulta vantaggioso distinguere tre generi di operazioni:

[α] Operazioni alle quali si attribuisce tutto o gran parte dell'onere delle elaborazioni e per le quali si assume una durata unitaria (tipicamente nel caso di elaborazioni numeriche queste sono le moltiplicazioni e le divisioni).

[β] Operazioni che richiedono tempi trascurabili rispetto alle precedenti (spesso si considerano tali le operazioni di indirizzamento rispetto a somme e sottrazioni oppure queste ultime rispetto alle moltiplicazioni e alle divisioni).

[γ] Operazioni che non si possono considerare elementari ma che devono essere ricondotte a quelle del genere [α]; tipiche operazioni considerate di questo genere sono le operazioni su numeri tanto grandi da richiedere più parole di memoria, le operazioni su polinomi, quelle su matrici, ...).

C47 b.04 Nelle considerazioni sulla complessità computazionale in genere si assume come unità di tempo la durata delle istruzioni del suddetto genere [α].
La diverse durate delle istruzioni sono tenute presenti solo quando si devono confrontare diverse apparecchiature o diverse implementazioni in relazione a un particolare tipo di esecuzione impegnativa e costosa.

C47 c. tipi di analisi della complessità [1]

C47 c.01 Definiamo ora uno schema orientativo al quale conviene fare riferimento per le analisi della complessità computazionale.

Per problema intendiamo una coppia $P = \langle D, f \rangle$, dove D denota l'insieme dei complessi di dati che determinano le istanze del problema stesso ed f una funzione che ha D come dominio.
Senza ledere la generalità, D può considerarsi un linguaggio ed ogni stringa $d \in D$ può vedersi come una precisa descrizione o come una opportuna codifica di una istanza del problema P.
Il codominio della funzione $f(D)$ costituisce l'insieme dei risultati del problema; anche $f(D)$ può considerarsi come un linguaggio.
Una risoluzione del problema P consiste in un procedimnto per il calcolo effettivo della f.
In molti casi f ha come codominio l'insieme dei valori booleani $\{\mathsf{false}, \mathsf{true}\}$; in questo caso si parla di **problema di decisione**.

C47 c.02 Consideriamo anche una macchina reale o realizzabile M che consenta di calcolare effettivamente la f. In genere il calcolo della f da parte della M si può effettuare servendosi di diversi algoritmi; denotiamo con $\boldsymbol{A}_{P,M}$ il loro insieme.

Se $A \in \boldsymbol{A}_{P,M}$ è uno particolare di tali algoritmi e $d \in D$ una codifica di una particolare istanza del problema, denotiamo con $S_{M,A}(d)$ la sequenza di operazioni che costituisce l'esecuzione di A relativa a d e con $M_A(d)$ denotiamo la soluzione $f(d)$ della relativa istanza di P in quanto concretamente ottenuta facendo operare la macchina M sotto il controllo (o governo) di A.
Con una macchina M sufficientemente realistica, un algoritmo A sufficientemente dettagliato si può pensare come un usuale modulo di programma.

C47 c.03 Le misure di complessità del calcolo di $M_A(d)$ sono opportune valutazioni numeriche sulle sequenze $S_{M,A}(d)$. Vediamo alcuni esempi.

La lunghezza di $S_{M,A}(d)$, cioè il numero dei passi richiesti per risolvere l'istanza d del problema, costituisce una semplice valutazione del tempo richiesto per ottenere la soluzione.

Può essere opportuno semplificare considerando, dei passi della sequenza, solo quelli relativi alle operazioni del suddetto tipo (α); per esempio per l'usuale prodotto di due matrici $n \times n$ può essere opportuno limitarsi a considerare le n^3 moltiplicazioni richieste.

Valutazioni più precise si possono ottenere enumerando separatamente i passi relativi alle operazioni giudicate diversamente impegnative e pesarli diversamente; per esempio tenere presente che per il suddetto prodotto di matrici, si effettuano n^3 moltiplicazioni ed $n^2(n-1)$ somme.

C47 c.04 Il numero dei registri di memoria utilizzati nei passi della $S_{M,A}(d)$ forniscono invece una valutazione della complessità spaziale.
Anche questa valutazione spesso può essere semplificata, in quanto vi sono poche sequenze, tabelle o altri gruppi di memorie che costituiscono la parte prevalente dei registri utilizzati; quindi lecito trascurare le rimanenti al fine di ottenere rapidamente una valutazione approssimata significativa

C47 c.05 Per quanto riguarda la complessità strutturale, valutazione che spesso può ridursi alla valutazione del lavoro umano richiesto dalla implementazione, si hanno linee di indagine meno definite: i parametri associati ad $S_{M,A}(d)$ che cercano di darne una misura significativa sono meno semplici e meno soddisfacenti.

Accade quindi che si hanno studi che si sviluppano con obiettivi e metodi piuttosto diversi. Da una parte si hanno studi teorici che fanno ricorso a nozioni piuttosto astratte come le gerarchie di funzioni subricorsive oppure si servono delle caratteristiche dei livelli di annidamento presenti in un programma scritto in un opportuno linguaggio di programmazione.

Dall'altra si hanno studi pragmatici concernenti i metodi di programmazione, l'ingegneria del software (programmazione modulare, strutturata, orientata agli oggetti, visuale, orientata al riutilizzo di componenti, ...), la messa a punto di ambienti per lo sviluppo dei programmi e la definizione di nuovi linguaggi di programmazione.

C47 c.06 Gli studi più rilevanti sul piano quantitativo hanno riguardato la complessità temporale.

In effetti per studiarla, come per la complessità spaziale, sono possibili buone semplificazioni e chiare definizioni degli scopi.

Per la gran parte dei problemi impegnativi, inoltre, il tempo di esecuzione costituisce la risorsa più carente e questo fa della complessità temporale la valutazione più influente e più richiesta.

Negli ultimi decenni è cresciuta esponenzialmente la disponibilità di dispositivi di memoria capaci, accessibili e sicuri e si prevede che questa tendenza possa proseguire; a questo proposito sono da ricordare, dopo i miglioramenti dei dischi rigidi l'adozione dei cosiddetti dischi a stato solido, e la facilità di accedere a memorie remote (cloud).
di conseguenza le preoccupazioni per la risorsa memoria negli ultimi decenni sono andate diminuendo.

In effetti non sono molti i problemi che richiedono una memoria da giudicare molto elevata relativamente alle disponibilità attuali e prevedibili.
Si tenga presente a questo proposito che attualmente è normale disporre di computers con miliardi di bytes (ottetti di bits) nella memoria centrale e su disco simulato da dispositivi a stato solido), mentre i dischi esterni dei comuni computers portatili dispongono di terabytes, di trilioni di bytes.

Le difficoltà per la elaborazione dei dati collegate alla memoria riguardano piuttosto la possibilità di disporre di grandi masse di dati empirici di buona qualità; quindi non si tratta di operare sugli algoritmo, ma sulla organizzazione della gestione dei dati nei diversi ambienti di lavoro e sulla adozione di standards per la codifica dei dati e per la loro strutturazione.

Attualmente dunque non è molto sentita la necessità di misurare e migliorare la complessità spaziale: non accade spesso di affrontare concretamente problemi che richiedano più della memoria in dotazione dei computers disponibili o, nel caso di obiettivi molto impegnativi, della memoria disponibile ai sistemi di milioni di processori operanti in parallelo.

C47 c.07 Molto maggiore è l'interesse per la complessità temporale. Ci troviamo di fronte a un gran numero di problemi che potrebbero essere risolti con grande utilità, per i quali sono disponibili considerevoli porzioni dell'insieme D dei dati idealmente richiesto, ma per i quali sono noti solo procedimenti algoritmici risolutivi che richiederebbero tempi di calcolo troppo elevati anche per i processori oggi disponibili o prevedibili a breve termine.

Talora sarebbero necessari tempi impraticabili anche per i processori che si possono prospettare in relazione alle miglior aspettative di crescita nei prossimi anni della potenza dei dispositivi elettronici.

In effetti gli studi sulla complessità temporale hanno consentito di migliorare in misura rilevante molti algoritmi tradizionali ed hanno reso praticabili elaborazioni di grande importanza.

C47 c.08 Vediamo ora cosa può dirsi sulla scelta della gamma delle istanze del problema nello studio della complessità temporale e spaziale degli algoritmi.

Questa dipende eminentemente dalle finalità specifiche degli studi, cioè dal contesto nel quale si vogliono mettere a punto gli algoritmi.

L'analisi probabilistica dettagliata di un algoritmo in relazione alla distribuzione delle sue istanze è stata portata avanti in un numero relativamente ristretto di casi, in quanto deve fare ricorso a tecniche enumerative assai complesse.
Per la presentazione di questi studi, dei quali è stato iniziatore Knuth, [*Greene, Knuth 1981*] rimandiamo al testo di *Sedgewick e Flajolet, 1996*.

C47 c.09 Nella maggior parte degli studi viene adottata la semplificazione consistente nel caratterizzare l'onerosità delle istanze $d \in D$ di un problema P con un unico parametro, tradizionalmente denotato con n, che può assumere solo valori interi positivi, e di considerare sostanzialmente equivalenti dal punto di vista della complessità le diverse istanze caratterizzate dallo stesso n.

Per molti problemi l'adozione di un unico parametro per l'onerosità delle istanze è pienamente naturale e giustificata.
Ad esempio nei problemi di ordinamento e di ricerca relativi a una sequenza di oggetti ordinabili come numeri e stringhe di lunghezze simili, è naturale assumere per n il numero dei componenti della sequenza data.
Similmente nei problemi concernenti matrici quadrate aventi un numero limitato di componenti nulle (moltiplicazione, soluzione di sistemi lineari, problemi di autovalori e autovettori) non vi sono alternative ad assegnare al numero delle righe e delle colonne delle matrici il ruolo di n.

C47 c.10 In altri tipi di problemi l'imposizione di un unico parametro per l'onerosità (ma si parla anche di dimensiome delle istanze) può costituire una forzatura. Per esempio per problemi sui grafi per la scelta del parametro si pone l'alternativa tra numero dei nodi e numero dei lati; per problemi riguardanti tabelle si ha l'alternativa tra numero di righe e numero di colonne.

La scelta di uno solo tra questi parametri può risultare fortemente condizionata dal fatto che i rimanenti possibili parametri dipendano da quello scelto in modo sufficientemente definito.
Per esempio per un problema sui grafi si può stabilire come parametro n il numero dei nodi, e quindi assumere che il numero dei lati sia vicino a $4\,n$ oppure a $n^2/3$; la prima assunzione è lecita quando si trattano grafi planari, la seconda quando si trattano grafi nei quali quasi tutti i nodi sono collegati a una grande parte degli altri.

Per molti problemi l'imposizione di un unico parametro dimensionale costituisce, in sostanza, una restrizione delle situazioni alle quali i risultati sono applicabili.

C47 c.11 Una considerazione a favore della "monodimensinalizzazione" delle istanze riguarda la possibilità di assumere come dimensione di una generica istanza di un problema, comunque complicata da descrivere, il numero di caratteri di una stringa mediante la quale tale istanza può essere espressa.
In particolare è interessante considerare il numero di bits necessari per precisare l'istanza alla macchina da utilizzare M.
Ancor più in particolare si ha il minimo numero di bits necessari alla suddetta precisazione. In questo modo si possono rendere relativamente inessenziali le distinzioni sulle modalità di codifica dei dati d'ingresso che talora possono preoccupare.

C47 c.12 La scelta di un unico parametro n di dimensionamento delle istanze in genere si accompagna alla decisione di studiare l'andamento asintotico di una misura di complessità espressa come dipendente da tale n. L'atteggiamento del privilegiare le situazioni asintotiche è adottato molto frequentemente.

Esso innanzi tutto consente di andare alla sostanza delle difficoltà dell'algoritmo, permettendo di trascurare effetti che si incontrano in poche situazioni di dimensioni ridotte e che quindi si possono considerare accidentali.

Esso inoltre dà maggior peso alle situazioni più impegnative; in effetti la elevata potenza degli odierni elaboratori consente di concentrare le preoccupazioni di chi progetta e implementa algoritmi sopra le situazioni più onerose e di considerare decisamente trascurabile le situazioni meno impegnative.

Spesso poi si ottengono andamenti asintotici delle funzioni di complessità nella sola variabile n molto chiari e significativi dai quali si possono ricavare utili considerazioni orientative anche sulle situazioni più complesse e realistiche.

All'atteggiamento di privilegiare le situazioni asintotiche si può obiettare di andare contro le ipotesi [EI3] ed [EI4] [b02].
Occorre però precisare che prestando attenzione agli andamenti asintotici non si vogliono considerare solo i valori enormemente elevati di n che imporrebbero di servirsi di registri multipli e di parallelismi di accesso ai banchi di memoria ben superiori a quelli di esteso utilizzo. Si vogliono piuttosto considerare valori elevati tanto da poter trascurare i contributi alla complessità secondari rispetto a quelli che prevalgono asintoticamente.

C47 c.13 Spesso è opportuno fare riferimento a due valori limit per n, n_1 ed n_2, individuati in modo approssimato come segue:

- n_1 esprime il valore di n al di sopra del quale possono essere empiricamente trascurati i contributi alla complessità che scompaiono asintoticamente;
- n_2 individua un valore tale che mantenendo n al di sotto di esso il sistema di calcolo utilizzabile dispone di registri singoli e di memorie complessive tali da rendere trattabili senza **rischio di overflows** le istanze del problema.

C47 c.14 Il privilegiare le situazioni asintotiche è giustificato solo se l'intervallo $[n_1 : n_2]$ è sufficientemente esteso, ovvero sse nel suddetto intervallo si può far cadere un insieme di istanze significativamente elevato in relazione alle esigenze applicative.

Questi due estremi vanno visti in relazione alle capacità dei sistemi correntemente disponibili e, per progetti impegnativi, in relazione alle prospettive di sviluppo degli strumenti nel breve e nel medio periodo.

Per quanto riguarda il limite inferiore in genere oggi si può essere piuttosto drastici: se per piccoli n l'andamento asintotico non prevale e non viene avvicinato, poco importa: si presume che i calcoli verranno comunque effettuati senza difficoltà.

Per quanto riguarda il limite superiore si possono distinguere tre classi di problemi.

[a] Si hanno algoritmi per i quali la gran parte dei casi di interesse pratico sono trattabili con elaboratori comuni.

[b] Alcune istanze sono trattabili con apparecchiature comuni e altre trattabili solo con macchine speciali; per questi problemi è importante tenere presenti le prospettive relative ai successivi 2-5 anni. Tenendo conto che per le apparecchiature utilizzabili ogni 18 mesi si ha all'incirca il raddoppio del rapporto prestazioni/prezzi, può essere opportuno pianificare di affrontare negli anni prossimi istanze relative a valori di n progressivamente crescenti.

[c] Si dispone di algoritmi per i quali n_2 è troppo piccolo per situazioni interessanti. In questi casi occorre chiarire se si potranno utilizzare gli algoritmi noti entro un certo numero di anni, oppure se si rendono necessarie forti riduzioni delle richieste che caratterizzano il problema.

C47 c.15 Accade comunque spesso che una soddisfacente misura di complessità sia fornita da una funzione che esprime l'andamento asintotico del tempo di calcolo richiesto o dello spazio di memoria occupato in dipendenza del solo parametro n.

Le funzioni asintotiche nella forma più semplice si limitano a segnalare l'ordine di grandezza del tempo di calcolo; valutazioni di complessità crescenti sono sono odate da

$$\mathbf{O}(\log(n)) \quad , \quad \mathbf{O}(n) \quad , \quad \mathbf{O}(n^h) \quad , \quad \mathbf{O}(k^n) \quad , \quad \mathbf{O}(n!) \quad , \quad \mathbf{O}(n^n) \quad , \quad ...,$$

Queste segnalazioni di complessità hanno il notevole vantaggio di fornire in termini estremamente semplici un'idea complessiva delle risorse richieste dall'algoritmo, idea che di fatto risulta adeguata nella maggior parte delle situazioni pratiche.
Con le precedenti espressioni nel parametro n si può inoltre dare una utile classificazione di gran parte degli algoritmi effettivamente adottati.

C47 d. tipi di analisi della complessità [2]

C47 d.01 Per analizzare un particolare algoritmo A relativo a un problema P ci si può chiedere, in relazione a un determinato insieme D di istanze di P, qual'è il dispendio medio di una risorsa, oppure qual'è il suo comportamento nel caso peggiore, cioè qual'è il massimo ammontare di una risorsa che gli può accadere di consumare.

Meno frequenti sono gli studi probabilistici sull'algoritmo, sia perché nettamente più impegnativi, sia perché in genere si rivela molto più problematico individuare una distribuzione di probabilità per le istanze del problema che da un lato rispecchi situazioni che si possono concretamente incontrare in un contesto applicativo e dall'altro consenta un trattamento quantitativo. con risultati utilizzabili.

C47 d.02 Un tipo particolarmente importante di analisi della complessità, ma in genere molto impegnativo, si pone come fine la dimostrazione di disporre di un **algoritmo ottimale**, cioè di un algoritmo che risulti tra i più efficienti relativamente al consumo di una specifica risorsa tra quelli dell'insieme disponibile $\boldsymbol{A}_P$.

Osserviamo esplicitamente che si è parlato di algoritmo ottimale e non di algoritmo ottimo. In effetti in genere di un algoritmo si possono trovare con facilità varianti poco rilevanti che consentono di risolvere lo stesso problema con una efficienza sostanzialmente uguale.
Notiamo anche che per taluni problemi si può avere un algoritmo ottimale per certi tipi di istanze (in particolare per certi valori del parametro dimensionale n) il quale è sopravanzato da un altro algoritmo per altri tipi di istanze.
La complessità (temporale o spaziale) di un algoritmo ottimale per un problema P si dice anche **complessità inerente** di P.

C47 d.03 Per la grande maggioranza dei problemi, però, non si è stati in grado di individuare un algoritmo ottimale. Più modestamente si è solo stabilito qualche limite inferiore per le misure di complessità (media, del caso peggiore o asintotica).

Peraltro anche una informazione sopra un limite inferiore spesso può costituire un elemento orientativa di rilevante utilità per orientare sulla complessità inerente del problema. Questo accade soprattutto quando è poco elevata la differenza tra la misura di complessità dei migliori algoritmi noti e il corrispondente limite inferiore.

C47 d.04 Un altro tipo di investigazione sulla complessità che può essere di buona utilità orientativa riguarda il **ricondurre la soluzione di un problema a quella di un problema precedente**.

Conoscendo le caratteristiche di un problema sufficientemente indagato P, di fronte a un nuovo problema $P' = \langle D', f' \rangle$ si tratta di mostrare che si riesce a risolvere P' utilizzando un algoritmo che sappiamo risolvere problema P servendosi di opportune trasformazioni di adattamento.

Per precisare meglio questo importante tipo di situazione, supponiamo che l'algoritmo A consenta di risolvere P fornendo un meccanismo costruttivo $\overline{f}$ che realizza la funzione $f \in \lceil\!\lceil\, D \longmapsto\!\triangleright\, R \,\rceil\!\rceil$, con $R := f(D)$ insieme dei risultati che costituiscono la risoluzione di P.
Quello che si può fare consiste nell'individuare due funzioni effettivamente costruibili $c \in \lceil\!\lceil\, D' \longmapsto D \,\rceil\!\rceil$ e $d \in \lceil\!\lceil\, R \longrightarrow R' \,\rceil\!\rceil$ con $R' := f'(D')$ insieme dei risultati che costituiscono la soluzione di P', funzioni tali che $c \circ \overline{f} \circ d$ costituisce una realizzazione della funzione f'.
La funzione c consente di **codificare una istanza di problema** P' mediante una istanza di P, mentre la d permette d **decodificare un risultato del problema** P ottenendo un risultato per P'.

$$\begin{array}{ccccc} & & f & & \\ & D & \longmapsto & R & \\ c & \uparrow & // & \downarrow & d \\ & D' & \longmapsto & R' & \\ & & f' & & \end{array}$$

C47 d.05 La riconduzione di un problema P' a un P sufficientemente noto può essere utile quando la messa a punto di c e d è abbastanza agevole, ossia quando presenta una bassa complessità strutturale.

In tal caso organizzare un procedimento risolutivo di P' avvalendosi di un algoritmo A_f noto può risultare vantaggioso, in quanto si può avere una buona garanzia di avere risultati utili in tempi contenuti.
Questa garanzia in molte circostanze applicative è giudicata importante, mentre si dà meno peso alla opportunità di individuare un procedimento risolutivo diretto di P' più efficiente.

Questo procedimento indiretto ($c \circ_{lr} f \circ_{lr} d$) potrebbe avere carattere provvisorio, in quanto uno studio approfondito di P' potrebbe condurre ad algoritmi specifici A' molto più efficienti e diretti. Un tale miglioramento dell'efficienza è più probabile nel caso di problemi P' con impatto maggiore e che presenta istanze continuamente rinnovate.

Occorre osservare che la provvisorietà di tante soluzioni algoritmiche non va giudicata in modo decisamente negativo: in uno scenario in continua rapida evoluzione come quello dell'elaborazione automatica dei dati la garanzia di ottenere soluzioni utili con procedimenti poco eleganti e poco efficienti in genere è giudicata preferibile alla prospettiva di un rilevante ritardo nella disponibilità di soluzioni.

Quando poi la somma delle complessità di c e d è sensibilmente inferiore a quella di A_f, l'algoritmo corrispondente alla funzione costruttiva $c \circ f \circ d$ presenta una complessità sostanzialmente non superiore a quella di A_f e quindi può essere utilizzato senza gravi preoccupazioni di poca eleganza o di bassa efficienza.

In effetti attualmente nell'ambito dell'ingegneria del software si dà molto peso alla **riutilizzabilità del software**, cioè alla possibilità di servirsi di meccanismi di ampia portata, anche se non della massima efficienza.

La riconduzione di un problema a uno noto può vedersi anche come tendenza a servirsi di pochi strumenti di alta versatilità.

C47 d.06 La riconduzione di un problema a uno noto può vedersi come elemento che consente di unificare questioni computazionali apparentemente distanti e quindi come contributo a economia di pensiero (oltre che di lavoro di programmazione).

In questo spirito vengono studiati procedimenti per la trasformazione di problemi di varia natura in problemi riguardanti oggetti "standard" come riconoscimento di linguaggi formali, calcolo di funzioni su interi, manipolazioni di strutture grafiche e valutazione di funzioni booleane.

Un atteggiamento di grande interesse consiste nel ricondursi a problemi riguardanti strutture matematiche più generali di quelle sulle quali certi problemi sono stati impostati.
Esempi notevoli sono dati da problemi riguardanti grafi nonorientati, strutture di incidenza [D64] e matrici i quali possono essere ricondotti a problemi su **matroidi** [D48].

Naturalmente quando si riconduce un algoritmo a uno di portata più generale può accadere di perdere in efficienza e in particolare di allontanarsi dalla ottimalità e dalla quasi ottimalità.

Per contro si possono ottenere una maggiore eleganza, cioè maggiore concisione nella impostazione, maggiore compattezza e maggiore versatilità dei programmi messi a punto nell'ottica della della generalità unita alla versatilità.

Una scelta tra le due tendenze, o un compromesso, nel breve periodo deve tener conto soprattutto delle circostanze specifiche, mentre in una ottica di maggior respiro, stante la crescita della strumentazione, può puntare a sistemi software di elevata adattabilità.

C47 d.07 Molti studi sugli algoritmi si pongono obiettivi più ridotti della complessità inerente, della quasi ottimalità e della elevata generalità. Su un piano a prima vista più modesto si collocano gli studi rivolti a individuare algoritmi sensibilmente più efficienti dei metodi tradizionali.
Negli anni recenti, con la crescente differenziazione e qualificazione delle tecniche della elaborazione automatica dei dati, è stato individuato un gran numero di algoritmi che migliorano i metodi tradizionali in misura decisiva, e talora sorprendente.

Anche se non si può parlare di quasi ottimalità (in quanto si è lontani dalle indicazioni note sui limiti inferiori, oppure in quanto non si dispone di alcuna valutazione su tali limiti) oggi si dispone di algoritmi molto ingegnosi che rispetto ai tradizionali presentano vantaggi di efficienza i quali sono stati determinanti perché si potessero affrontare problemi e tematiche applicative precedentemente giudicate intrattabili.

Questa cresciuta portata del complesso degli algoritmi di base è portato vantaggio in particolare alla possibilitdi costruire sistemi per affrontare problematiche articolate (i cosiddetti "sistemi multitasking", spinti anche dalla miniaturizzazione e dalla telematica) assemblando agevolmente e in tempi contenuti molteplici efficienti algoritmi di base.

C47 d.08 Un esempio estremamente semplice di problema per il quale il passaggio dal procedimento risolutivo più intuitivo a uno più studiato approfonditammente comporta un sostanziale abbassamento della complessità riguarda la ricerca di un elemento in una lista di oggetti totalmente ordinati.

Problema: Data una sequenza ordinata di oggetti distinti che scriviamo $\langle x_1 < x_2 < ... < x_n \rangle$, ad esempio una lista di numeri razionali, si chiede di individuare una posizione nella quale un nuovo oggetto $\overline{x}$ della stessa natura si possa inserire o si possa già trovare.

Un algoritmo risolutore intuitivo e semplice da implementare è il seguente.

Algoritmo: (**ricerca sequenziale**) Si procede a confrontare x con x_1, x_2, ... x_i, ... fino a che si trova x_j tale che $x \leq x_j$. Se si trova $x = x_j$ si conclude che l'oggetto cercato è nella posizione j-esima; se si trova $x < x_j$ si conclude che x può inserirsi appena prima di x_j. Se non si trova uno x_j come quello richiesto si conclude che x può porsi in coda alla sequenza ∎

Chiaramente questo algoritmo richiede in media $\left\lfloor \frac{n}{2} \right\rfloor$ confronti, mentre nel caso peggiore richiede n confonti. Il comportamento asintotico della complessità dell'algoritmo è comunque del tipo $\mathbf{O}(n)$.

C47 d.09 Un algoritmo leggermente meno intuitivo è il seguente:

Algoritmo: (**ricerca dicotomica**) Si confronta x con la cosiddetta **componente intermedia di una sequenza data**, che scriviamo x_{n_1} con $n_1 := \lceil n/2 \rceil$; x_{n_1} è una delle al più due componenti più vicine alla metà della sequenza (se n è dispari è l'unica, se n è pari l'altra è $x_{(n+1)/2}$).
Se $x \leq x_{n_1}$ si è ridotti alla ricerca sulla sottosequenza $\langle x_1 < x_2 < ... < x_{n_1} \rangle$; in caso contrario si deve effettuare la ricerca sulla sottosequenza $\langle x_{n_1+1} < ... < x_n \rangle$. Con un confronto si è quindi ridotta circa della metà la sequenza su cui ricercare.

Ciascuno dei passi successivi riguarda il confronto di x con la componente intermedia della sequenza rimasta ∎

Nell'ipotesi che un confronto non consenta di individuare una coincidenza $x = x_j$, ovvero nell'ipotesi che la coincidenza sia molto improbabile, il procedimento ha termine quando rimane un solo elemento con cui confrontare x; questo accade dopo $\lceil \log_2(n) \rceil$ passi. Questa valutazione di complessità temporale riguarda sia un comportamento medio sia il caso peggiore.

C47 d.10 L'algoritmo di ricerca dicotomica, rispetto all'algoritmo di ricerca sequenziale, presenta una complessità strutturale un po' maggiore ma una complessità temporale del tipo $\mathbf{O}(\log_2(n))$ sostanzialmente inferiore a quella del tipo $\mathbf{O}(n)$.

Dato che i problemi di ricerca si presentano molto frequentemente nella elaborazione dei dati e possono riguardare molte migliaia o anche molti milioni di oggetti da mantenee ordinati, l'algoritmo dicotomico è nettamente da preferire.
Ogni passo dell'algoritmo dicotomico potrebbe avere anche una durata 5 volte maggiore di un passo dell'algoritmo sequenziale; dovremmo quindi confrontare gli $s := n/2$ passi elementari dell'algoritmo sequenziale contro i $d := 5\log_2(n)$ passi equivalenti del dicotomico.

Se $n = 32$ si hanno $s = 16$ passi contro $d = 5 \cdot 5 = 25$, ma già per $n = 64$ si hanno 32 passi contro 30 e per n maggiori il vantaggio del procedimento dicotomico si fa vistoso: per $n = 1024$ siamo a 512 contro 50, per $n = 2^{20} \approx 10^6$ siamo a $s = 0.510^6$ contro $d = 100$ e per $n = 2^{30}$ abbiamo $s = 0.5\,10^9$ contro $d = 150$. Se ne conclude che solo per ricerche su insiemi molto piccoli ha senso servirsi dell'algoritmo strutturalmente meno impegnativo.

Osserviamo anche che l'algoritmo dicotomico si dimostra essere ottimale.

Per rendersi conto di questo si può pensare che ogni algoritmo che risolve il problema della ricerca deve individuare una posizione tra le $n + 1$ possibili determinate dalle n componenti della sequenza data, servendosi di confronti, cioè di una operazione che fornisce una informazione binaria.

La posizione richiesta è individuata naturalmente dalla scrittura binaria di un intero appartenente a $[0 : n]$; questa richiede $\lfloor \log_2(n+1) \rfloor$ bits e quindi non si può avere una complessità inferiore a una del genere $\mathbf{O}(\log_2(n))$.

C47 d.11 **Problema**: Calcolare il valore del polinomio $P(x) = a_0 + a_1x + ... + a_nx^n$.

Algoritmo: (**metodo do Horner**) La valutazione richiesta è determinata dalla formula

$$P(x) = (...((a_nx + a_{n-1}x + a_{n-2})x + ... + a_1)x + a_0 \ .$$

essa procede dal contenuto delle parentesi più interne con una sequenza di n prodotti ciascuno seguito da una somma ∎

In generale, cioè in mancanza di numerosi coefficienti nulli e di forti correlazioni tra i coefficienti, l'algoritmo richiede n moltiplicazioni (ed n somme ed n manovre su indici) e non è difficile rendersi conto che l'algoritmo di Horner risulta ottimale sia in media, che nelle condizioni peggiori.

C47 d.12 **Problema**: Calcolare il prodotto di due matrici 2×2

$$\begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix} := \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \cdot \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \ .$$

Algoritmo: (**algoritmo di Strassen**) Si utilizzano le seguenti formule

$$t_1 := (a_{1,1} + a_{2,2})(b_{1,1} + b_{2,2}) \ , \qquad t_2 := (a_{2,1} + a_{2,2})b_{1,1} \ , \qquad t_3 := a_{1,1}(b_{1,2} - b_{2,2}) \ ,$$

$$t_4 := a_{2,2}(b_{2,1} - b_{1,1}) \ , \qquad t_5 := (a_{1,1} + a_{1,2})b_{2,2} \ , \qquad t_6 := (a_{2,1} - a_{1,1})(b_{1,1} + b_{1,2}) \ ,$$

$$t_7 := (a_{1,2} - a_{2,2})(b_{2,1} + b_{2,2} \ ,$$

$$c_{1,1} := t_1 + t_4 - t_5 + t_7 \ , \qquad c_{1,2} := t_3 + t_5 \ , \qquad c_{2,1} := t_2 + t_4 \ , \qquad c_{2,1} := t_1 + t_3 - t_2 + t_6 \ \blacksquare$$

L'algoritmo di Strassen è molto più macchinoso di quello basato sulle formule definitorie $c_{i,k} = a_{i,1}\, b_{1,k} + a_{i,2}\, b_{2,k}$, ma richiede 7 moltiplicazioni invece delle 8 del tradizionale.
Uno schema di calcolo analogo e generale si può trovare per il prodotto di due matrici $n \times n$ e richiede $n^{\log_2(7)} = n^{2,80735492432547\ldots}$ moltiplicazioni invece delle n^3 del metodo derivante dalla definizione di prodotto tra matrici.
Con n elevato l'algoritmo di Strassen presenta vantaggi rilevanti, e anche il prodotto di matrici è una manovra che si incontra molto spesso e che presenta grandissimo interesse per tutti i settori della produzione, della tecnologia e della ricerca.

Conviene segnalare che l'algoritmo di Strassen non è ottimale, se ne conoscono di migliori. Esso però ha avuto il grande merito di mostrare che si potevano trovare approcci non intuitivi ma sensibilmente vantaggiosi anche per problemi di calcolo molto radicati nella tradizione dando un forte impulso alla ricerca di algoritmi innovativi.

C47 e. problemi polinomiali e problemi intrattabili

C47 e.01 Un algoritmo si dice di complessità polinomialmente limitata o in breve **algoritmo polinomiale**, sse la sua complessità è esprimibile come $\mathbf{O}(p(n))$ con $p(n)$ polinomio nel parametro dimensionale n.
La classe di tali algoritmi si denota con $\mathcal{P}$. I restanti algoritmi, cioè quelli la cui complessità non si può limitare con una espressione n^h si dicono **algoritmi più che polinomiali**.

Tra gli algoritmi polinomiali risultano particolarmente vantaggiosi quelli limitati da funzioni logaritmiche, del tipo $\mathbf{O}(\log(n))$ o $\mathbf{O}(\log(n)^k)$ con $k \in \mathbb{P}$; questi vengono detti **algoritmi logaritmici**. Si tratta di algoritmi che, come quello di ricerca dicotomica, consentono di operare rapidamente anche su grandissime quantità di dati.

Tra i restanti polinomiali talora è molto importante riuscire a distinguere tra gli **algoritmi lineari**, del livello $\mathbf{O}(n)$, gli **algoritmi quadratici**, del livello $\mathbf{O}(n^2)$, gli **algoritmi cubici**, del livello $\mathbf{O}(n^3)$, e così via.

Spesso si fa riferimento a livelli di complessità di algoritmi che si collocano tra i precedenti come il livello $n^{\log_2(7)}$

C47 e.02 In effetti in queste classi si collocano molti algoritmi di grande interesse pratico: in particolare algoritmi sui grafi utilizzabili in numerose applicazioni [v.o.] e algoritmi che svolgono i ruoli centrali nei compilatori per i linguaggi di programmazione e nei sistemi per la gestione di basi dati.

Le distinzioni tra i sopra accennati livelli di complessità temporale corrispondono a notevoli differenze in termini di costi per elaborazioni che spesso sono effettuate intensivamente.

Accade quindi che vengano dedicati molte energie allo studio di algoritmi che consentano di passare da una classe del tipo $\mathbf{O}(n^k)$ ad una classe del tipo $\mathbf{O}(n^{k-\delta})$, anche per differenze δ sensibilmente minori di 1.

In particolare tra gli algoritmi lineari si collocano quelli riguardanti l'analisi o la trasformazione di stringhe mediante digrafi di transizione, mentre tra i quadratici e i cubici si collocano gli algoritmi che consentono di operare su alberi sintattici al fine di comprendere il significato operativo delle espressioni e dei costrutti più importanti dei linguaggi di programmazione.
A questa differenza nell'esponente di n per i corrispondenti livelli di complessità $\mathbf{O}(n^k)$ esprime significativamente il diverso impegno dei due suddetti tipi di manipolazioni di stringhe.

C47 e.03 Anche tra gli algoritmi più che polinomiali è necessario fare distinzioni.

Si parla quindi di **algoritmi esponenziali** quando la complessità è di un livello $\mathbf{O}(a^n)$ per qualche $a \in (1, +\infty)$, di **algoritmi fattoriali** per livelli $\mathbf{O}(n!)$ e di **algoritmi più che fattoriali** per livelli ancora più elevati.

Tra questi ultimi si può considerare la generazione di tutti i quadrati latini dei vari ordini [D63]. Dato che il numero dei quadrati (standard o generali) cresce ben più di $n!$, è sicuro che la loro generazione costituisce un problema più che fattoriale.
Considerazioni analoghe si possono svolgere per i problemi di generazione delle successioni di tutte la classi di isomorfismo di numerose strutture algebriche, combinatorie e topologiche.

Le qualifiche di complessità assegnate agli algoritmi si trasmettono in modo prevedibile anche ai problemi.
Per esempio un problema si dice cubico se si conosce un suo algoritmo risolutivo del tipo $\mathbf{O}(n^k)$ con $k \approx 3$ e non si conosce alcun algoritmo limitato da una potenza di n sensibilmente inferiore

C47 e.04 Per gli algoritmi per i quali si riesce a riferire la complessità ad un solo parametro si viene dunque a individuare una gerarchia che permette di avere una visione d'insieme relativamente semplice e di grande utilità pratica dei procedimenti computazionali.

La distinzione più marcata che si riscontra in questa gerarchia riguarda gli algoritmi polinomiali da una parte e quelli di complessità superiore dall'altra.

In effetti accade che un problema di cui si conosce un algoritmo polinomiale si riesce a trattare in una gamma di istanze abbastanza estesa e sostanzialmente si riesce a dominare in modo piuttosto sistematico con automatismi accettabili, anche in termini della evoluzione delle applicazioni realizzabili consentita dalla crescita delle tecnologie di base.

Si potrebbe obiettare che quando si ha una limitazione $\mathbf{O}(p(n))$ con $p(n)$ polinomio di grado moto elevato si rende necessario limitare la dimensione n delle istanze effettivamente trattabili e in tali condizioni potrebbe risultare preferibile un buon algoritmo esponenziale che potrebbe essere più facilmente implementabile e adattabile.

Tuttavia non si conoscono algoritmi di rilevante interesse aventi la complessità limitata da polinomi di grado sostanzialmente superiore al quinto.

Per gli algoritmi più che polinomiali quindi in genere si adotta il termine **algoritmi intrattabili** e si deve prestare molta attenzione ai **problemi intrattabili**.

C47 e.05 L'importanza della distinzione tra problemi polinomiali e più che polinomiali è stata introdotta da *Alan Cobham 1964* ed *Edmonds 1965*. Tale distinzione trova un forte supporto nel fatto, scoperto da Cobham, che il carattere polinomiale di un algoritmo è indipendente dalla macchina che si utilizza e dalla scelta del parametro dimensionale per le istanze, naturalmente quando si prendano in considerazione problemi significativi proposti entro circostanze ragionevoli.

Il termine intrattabile è dovuto al fatto che con questi algoritmi si deve prestare molta attenzione alle istanze da sottoporre a elaborazione automatica per evitare di incorrere in una delle cosiddette **esplosioni combinatorie**.
Si tratta di situazioni nelle quali cambiando poco i dati di ingresso si perde la possibilità di avere soluzioni in tempi praticabili.
Si pensi ad esempio alla generazione dei quadrati latini standard al crescere del loro ordine n. La cosa è fattibile ormai facilmente fino ad $n = 7$, ma il problema per $n = 8$ diventa molto pesante anche per le più potenti tra le macchine attuali e per $n = 9$ diventa irragionevole affrontarlo in quanto richiederebbe tempi misurabili in migliaia di anni.

Occorre osservare che dovendosi affrontare una elaborazione di questa portata potrebbe essere opportuno aspettare la disponibilità di macchine molto più potenti delle attuali prima di avviare le elaborazioni.
La crescita esponenziale delle prestazioni dei microprocessori verificatasi negli ultimi vent'anni (raddoppio del rapporto prestazioni/prezzi ogni 18 mesi) e la ragionevole prospettiva che una crescita comparabile prosegua nei prossimi venti anni potrebbero indurre a posporre l'avvio di un calcolo massiccio di una decina d'anni con la previsione di ottenere i risultati molto prima della aspettata conclusione di una elaborazione avviata con le apparecchiature odierne.

C47 f. problemi NP

C47 f.01 Vogliamo ora studiare una classe di problemi di grande interesse teorico ed applicativo che vengono collocati tra i polinomiali e gli intrattabili.

Si tratta di problemi, molti dei quali di grande importanza, che possono essere risolti da algoritmi esponenziali, ma nonostante siano stati studiati molto approfonditamente, non si è riusciti a dimostrare che la loro complessità inerente è esponenziale e quindi che non possono essere risolti da un algoritmo polinomiale. Rimane dunque la speranza che essi non siano problemi intrattabili.

I problemi di questo genere inoltre sono caratterizzati dal fatto che potrebbero essere risolti in tempi polinomiali se si potesse disporre di apparecchiature in grado di operare con un esteso parallelismo.

C47 f.02 Prima di dare definizioni precise diamo due esempi di tali problemi.

Problema: (**problema della cricca o del sottografo completo**) Dato un grafo nonorientato generico $G = \langle Q, U\rangle$ di n nodi e un intero positivo k inferiore ad n, stabilire se G possiede un sottografo completo (o cricca) di k nodi.

Questo problema viene denotato con $CLIQ$.

Mostriamo un algoritmo piuttosto semplice che consente di risolvere $CLIQ$ in tempi esponenziali nel parametro dimensionale n.

Primariamente si organizza una corsa per generare tutti i possibili k-sottoinsiemi di Q. Quindi per ciascuno di tali sottoinsiemi si stabilisce se si tratta di una cricca, cioè di un grafo completo $\boldsymbol{K}_k$. Questa decisione richiede il controllo della presenza in U di $k(k-1)/2$ lati e quindi ha un costo del tipo $\mathbf{O}(k^2)$; essendo $k < n$ si ha un limite del tipo $\mathbf{O}(n^2)$. Per la generazione dei $\binom{n}{k}$ sottoinsiemi è sufficiente un tempo esponenziale, in quanto il loro numero è inferiore al numero 2^n di tutti i sottoinsiemi di Q. Complessivamente quindi basta un tempo del tipo $\mathbf{O}(n^2\,2^n)$, cioè del tipo $\mathbf{O}(a^n)$, ovvero si ha un algoritmo di complessità esponenziale.

C47 f.03 Prima di introdurre il secondo problema richiamiamo alcune nozioni riguardanti le espressioni booleane [B60g].

Per **variabile booleana** si intende una variabile che può assumere i valori false e true, o equivalentemente, i valori 0 ed 1.

Per **espressione booleana** nelle variabili x_1, x_2, ..., x_n si intende una espressione ben formata nella quale possono intervenire le suddette variabili e gli operatori di prodotto booleano o AND, di somma booleana o OR e di complementazione. Questi operatori verranno denotati risp. con "$\cdot$" e "$\dot{+}$" infissi e con " $'$ " suffisso.

Per **espressione booleana in forma normale congiuntiva**, in breve **espressione BFNC**, si intende una espressione booleana data da un prodotto di somme nelle quali compaiono variabili semplici e complementate.
In una di queste somme non possono comparire insieme una variabile e la sua complementata; infatti $x_i \dot{+} x_i' \dot{+} ... = 1$ e quindi una tale somma non influisce in quanto fornisce al prodotto complessivo un fattore che può essere eliminato dall'espressione.

Un esempio di espressione booleana nelle variabili x e y è $(xy' \dot{+} x'y)$; esempi di espressioni BFNC sono $(x \dot{+} y') \cdot (x' \dot{+} y') \cdot (x' \dot{+} y' \dot{+} z)$ e $(x \dot{+} y) \cdot (x' \dot{+} y') \cdot y$.

C47 f.04 Una espressione booleana in n variabili individua una **funzione di verità n-variata**, cioè una funzione del genere $\lceil \{0,1\}^n \longmapsto \{0,1\} \rceil$: infatti ogni assegnazione di valori booleani alle sue variabili fornisce un valore booleano.

Una espressione booleana si dice **espressione booleana soddisfacibile** sse esiste una assegnazione di valori booleani alle sue variabili che la rende vera, ovvero sse la funzione di verità associata non si riduce alla costante false.

Per esempio la $xy' \dot{+} x'y$ è soddisfacibile in quanto assegnando $x := 1$ e $y := 0$ fornisce il valore 1, mentre la $(x \dot{+} y') \cdot (x' \dot{+} y') \cdot (x' \dot{+} y' \dot{+} z)$ viene soddisfatta ponendo $x := 1$, $y := 0$ e $z := 1$.
Non è invece soddisfacibile la $x \cdot (x' \dot{+} y) \cdot y'$.

La soddisfacibilità o meno delle espressioni booleane semplici come l'ultima si può verificare anche con semplici manipolazioni simboliche; la cosa diventa però sempre più onerosa quando si trattano espressioni via via più estese.

C47 f.05 Problema: (**problema della soddisfacibilità delle espressioni BFNC**) :
Data una espressione BFNC, stabilire se è soddisfacibile.

Questo problema si denota con SAT.

Esso si risolve con un semplice algoritmo esponenziale consistente nel procedere alla generazione di tutte le 2^n possibili assegnazioni di valori alle n variabili booleane della espressione data e nel calcolo del valore fornito dall'espressione per ciascuna assegnazione.
In effetti il calcolo di ciascuna espressione BFNC può farsi rapidamente: si ha il valore 1 sse tutti i fattori danno 1 ed un fattore fornisce 1 sse almeno uno dei suoi addendi vale 1. Quindi la sua complessità è del tipo $\mathbf{O}(2^n)$.
Non si conosce invece alcun algoritmo polinomiale che consenta di risolvere SAT.

C47 f.06 I due precedenti problemi possono invece essere risolti in tempi ragionevoli da un sistema di calcolo in grado di far procedere in parallelo l'esame di diverse situazioni.

Apparecchiature del genere sono le macchine di Turing nondeterministiche. Invece di riferirsi a queste macchine costituite da componenti molto semplici come avviene nelle trattazioni prevalentemente teoriche, per argomentare sulla complessità è preferibile riferirsi a un altro modello di sistema di calcolo chiamato **multiprocessore ad albero**.

Tale sistema è da pensarsi costituito da molte repliche di un processore evoluto utilizzabile mediante programmi scritti in un usuale linguaggio di programmazione procedurale e resi in grado di effettuare la prestazione spaciale che segue.
Nel corso di una elaborazione un processore si può trovare di fronte alla necessità di analizzare situazioni alternative o di costruire strutture di dati alternative.
In questo caso esso può avvalersi della prestazione speciale consistente nell'attivare altri processori del sistema che in quel momento non siano impegnati perché portino avanti ciascuno una delle analisi o delle costruzioni alternative.
In uno stadio successivo i processori incaricati delle attività parallele possono rendere disponibili i loro risultati al processore che li ha attivati e porsi in attesa di nuovi incarichi dello stesso genere.

Un sistema come il precedente è il corrispondente evoluto, nel senso della ricchezza dei circuiti operativi dei quali si può avvalere, di una macchina di Turing deterministica.
Il complesso delle operazioni elementari eseguite in una sua elaborazione si può disporre secondo una arborescenza la quale presenta una diramazione in corrispondenza di ogni richiesta della prestazione speciale.

C47 f.07 Supponendo possibile di avvalersi di un multiprocessore ad albero, è abbastanza facile prospettare algoritmi che risolvono in tempi polinomiali i problemi descritti in precedenza.
.....
.....

C47 f.08 Seguendo Edmonds diciamo **problema** $\mathcal{NP}$ un problema che si sa risolvere mediante una macchina deterministica con un algoritmo esponenziale, che non si è dimostrato avere complessità inerente esponenziale e che si sa risolvere mediante una macchina nondeterministica in tempi polinomiali.
Denotiamo con $\mathcal{NP}$ l'insieme di questi problemi.

I discorsi fatti in precedenza dicono che $CLIQ$ e SAT sono problemi dell'insieme $\mathcal{NP}$.
Servendosi della nozione di multiprocessore ad albero risulta poi abbastanza agevole prospettare algoritmi che consentono di risolvere in tempi polinomiali una grandissima varietà di problemi.
In effetti attualmente si sanno assegnare ad $\mathcal{NP}$ molte migliaia di problemi riguardanti questioni computazionali di svariATI generi.

C47 g. problemi NP-completi

C47 g.01 Un fondamentale passo avanti sul problema del confine tra problemi polinomiali ed $\mathcal{NP}$ fu compiuto da *Cook 1971*, il quale riuscì a dimostrare il seguente fatto.

Teorema teorema di Cook
Ogni problema $\mathcal{NP}$ può essere ridotto polinomialmente al problema SAT.

Quindi se si trovasse un algoritmo che consente di risolvere con tempi polinomiali SAT si potrebbero risolvere in tempi polinomiali tutti i problemi in $\mathcal{NP}$.

C47 g.02 Successivamente *Karp 1972* riuscì a provare con un procedimento di portata generale che una ventina di altri problemi godono la proprietà secondo la quale ogni altro problema di $\mathcal{NP}$ può essere ricondotto ad essi con procedimenti polinomiali. Alcuni di questi hanno grande importanza applicativa.

Accade inoltre che la relazione tra problemi di $\mathcal{NP}$ consistente nella riconducibilità polinomiale a un problema polinomialmente equivalente a SAT è una relazione transitiva, come mostra il diagramma funzionale seguente

$$\begin{array}{ccccc} & \overset{c_1}{\longleftarrow} & & \overset{c_2}{\longleftarrow} & \\ A & & A_1 & & A_2 \\ \downarrow & & \uparrow & & \uparrow \\ f_S & & f_1 & & f_2 \\ \downarrow & & \uparrow & & \uparrow \\ B & \underset{d_1}{\longmapsto} & B_1 & \underset{d_2}{\longmapsto} & B_2 \end{array}$$

Nel diagramma f_S rappresenta una funzione che risolve il problema SAT le cui istanze sono formalizzate in enunciati costituenti A e i cui sono rappresentati da elementi di B; f_1 ed f_2 rappresentano analoghe funzioni risolutive, risp., per i problemi P_1 e P_2; c_1 e c_2 denotano funzioni di codifica in tempi polinomiali, risp., dai dati di P_1 ai dati di SAT e dai dati di P_2 ai dati di P_1; d_1 e d_2 denotano funzioni di decodifica in tempi polinomiali, risp., dai risultati di SAT ai risultati di P_1 e dai risultati di P_1 ai risultati di P_2.

Quindi ogni problema in $\mathcal{NP}$ si può ricondurre polinomialmente a ciascuno dei problemi considerati da Cook e Karp e questi possono ricondursi polinomialmente l'uno all'altro. In particolare SAT può essere ricondotto polinomialmente a $CLIQ$.

C47 g.03 Karp fu quindi indotto a introdurre la sottoclasse di $\mathcal{NP}$ comprendente tutti i problemi ai quali possono essere ricondotti polinomialmente tutti i problemi $\mathcal{NP}$; essa viene detta **classe dei problemi $\mathcal{NP}$-completi**.

È ragionevole ritenere che in questa classe si trovino i problemi $\mathcal{NP}$ più impegnativi, in quanto in grado di consentire la soluzione di moltissimi altri problemi.

Karp giunse quindi a porre in evidenza il fatto che se si riuscisse a trovare un algoritmo che risolve polinomialmente uno qualsiasi dei problemi $\mathcal{NP}$-completi, si avrebbe la possibilità di rendere trattabili, cioè di risolvere in tempi polinomiali, una grande quantità di problemi computazionali.

Si pone quindi il problema che si usa denotare con la scrittura $\mathcal{P} = \mathcal{NP}?$, riguardante il dilemma se sia $\mathcal{P} \subset \mathcal{NP}$ oppure se esista un modo di risolvere tutti i problemi $\mathcal{NP}$ in tempi polionomiali.

Questo importante problema è tuttora aperto, ma è accaduto che negli anni successivi ai lavori seminali di Cook e Karp si sia sviluppata una vivacissima ricerca sugli algoritmi che ha consentito di collocare nella classe dei problemi $\mathcal{NP}$ molte migliaia di problemi computazionali.

Naturalmente per ciascuno di questi problemi si è cercato un algoritmo polinomiale, senza tuttavia riuscire a trovarne alcuno.

C47 g.04 Questa situazione va a sostegno della seguente congettura:

Cong.: (**congettura di Karp**) $\mathcal{P} \subset \mathcal{NP}$.

Infatti si può ritenere assai improbabile che si possa trovare un algoritmo polinomiale per un problema $\mathcal{NP}$ dopo tutti i tentativi fatti sui moltissimi problemi $\mathcal{NP}$.

Presentiamo ora un elenco dei problemi $\mathcal{NP}$-completi

C47 g.05 **Problema**: Dato un grafo individuare una sua cricca massimale.

C47 g.06 **Problema**: Dato un grafo, stabilire se contiene un circuito o un cammino Hamiltoniano.

C47 g.07 **Problema**: Dato un digrafo, stabilire se contiene un circuito o un cammino Hamiltoniano.

C47 g.08 **Problema**: (**problema del commesso viaggiatore)** Data una rete, cioè un grafo con lati dotati di un peso, individuare un circuito Hamiltoniano di peso minimo.

C47 g.09 **Problema**: Dato un grafo, individuare un suo insieme dominante, cioè un insieme di suoi nodi che incidono in tutti i suoi spigoli.

C47 g.10 **Problema**: Dati un grafo e un intero k minore o uguale al suo ordine, cioè al numero dei suoi nodi, stabilire se esso contiene un insieme dominante formato da k nodi.

C47 g.11 **Problema**: Dato un grafo e un intero k minore o uguale al suo ordine, stabilire se esso possiede una k-colorazione.

C47 g.12 Dato un digrafo con circuiti, individuare un suo insieme minimale di nodi di feedback, cioè un insieme di nodi la cui eliminazione rende il digrafo aciclico.

C47 g.13 **Problema**: Dati un digrafo con circuiti e un intero k minore o uguale al suo ordine, stabilire se esso contiene un sottoinsieme di k nodi di feedback.

C47 g.14 **Problema**: Dato un digrafo con circuiti, individuare un suo insieme minimale di archi di feedback, cioè un insieme di archi la cui eliminazione rende il digrafo aciclico.

C47 g.15 Dati una rete, cioè un grafo con gli spigoli pesati, un sottoinsieme S del suo insieme dei nodi ed un numero positivo r, stabilire se è possibile porre in connessione tutti i nodi di S con un insieme di lati di peso totale non superiore ad r.

Ricordiamo che la sottoforesta individuata da questi lati si dice **foresta di Steiner**.

C47 g.16 Dati una rete e un sottoinsieme S del suo insieme dei nodi, individuare una foresta di Steiner per S di peso minimale.

Si noti che nel caso particolare in cui S contiene tutti i nodi del grafo il problema si riduce a quello della sottoforesta ricoprente minimale risolvibile con il classico algoritmo greedy [D48h] in tempi quadratici nell'ordine del grafo.

C47 g.17 Dati un insieme di attività, i tempi necessari allo svolgimento di ciascuna di esse, le date alle quali le attività si vogliono concluse, le penali da pagare nel caso di mancato rispetto di ciascuna scadenza ed un numero positivo r, stabilire se è possibile fissare un calendario delle attività in modo che la somma delle penalità da pagare per i lavori fuori termine non superi r.

C47 g.18 Problema: (**problema dell'assegnamento tridimensionale**): Dati un array tridimensionale binario e un intero positivo k, individuare un insieme di k posizioni nell'array con entrate uguali ad 1 tale che nessuna coppia di posizioni risulti allineata.

C47 h. convivere con i problemi intrattabili

C47 h.01 Per i problemi chiamati intrattabili, in genere, si rende necessario trovare approcci nuovi. Come è accaduto nel passato, alcuni avanzamenti delle conoscenze generali, matematico-computazionali, nei loro settori possono essere più determinanti dei pur spettacolari progressi tecnologici.

C47 h.02 Per molti problemi, tuttavia, c'è da pensare che sia più ragionevole ridurre le esigenze computazionali.

Per certe applicazioni può rendersi necessario rinunciarea modelli molto raffinati e ripiegare su modelli più approssimativi.

Per alcune attività di ottimizzazione può essere opportuno accontentarsi di situazioni sub-ottimali.

Per varie costruzioni può essere necessario ripiegare su procedimenti euristici.

In talune problematiche possono rendersi necessari i cosiddetti **algoritmi probabilistici**, algoritmi cioè che forniscono risultati garantiti solo al 99.8%, ma ottenibili in tempi contenuti, enormemente inferiori a quelli prevedibili per avere un risultato sicuro.

Altre strade che oggi destano sempre maggiore interesse sono quelle relative alla adozione di srumenti harware che consentono di effettuare elaborazioni massicciamente parallele.

Testo fruibile in https://www.mi.imati.cnr.it/alberto/ e https://arm.mi.imati.cnr.it/Matexp/matexp_main.php