Butlletí de la Societat Catalana de Matemàtiques
Vol. 27, núm. 1, 2012. Pàg. 39–62. DOI: 10.2436/20.2002.01.41

de la computació

Elitza Maneva

La ciència de la computació és tant sobre ordinadors com l'astronomia és sobre telescopis.

Edsger Dijkstra

Resum: El problema «P *versus* NP» és un dels set Problemes del Mil·lenni de l'Institut Clay de Matemàtiques, la solució del qual estaria premiada amb un milió de dòlars. En aquest article presentem de manera divulgativa el problema i els seus orígens, donant pel camí exemples de problemes computacionals de diferents nivells de dificultat, alguns algoritmes no trivials, la definició de màquina de Turing —el model matemàtic d'ordinador— i el concepte de reducció polinòmica entre problemes. La part més avançada de l'article presenta una demostració del teorema de Razborov sobre circuits monòtons de l'any 1985 que resol un cas especial de la conjectura. També donem una traducció al català d'una carta de Gödel a Von Neumann de l'any 1956 que es va descobrir l'any 1988 i que es pot considerar com la primera formulació per escrit del problema «P *versus* NP».

Paraules clau: complexitat computacional, algoritme, màquina de Turing, circuit.

Classificació MSC2010: 68-02, 68Q15, 68Q17, 68Q05.

1 Introducció

En el context de la vertiginosa velocitat a la qual avança la tecnologia informàtica, i la magnitud del seu efecte sobre la nostra civilització, sovint sembla un repte insuperable haver de convèncer algú de la rellevància d'un problema teòric i fonamental de la ciència de la computació, encara que sigui tan bàsic com el problema «P *versus* NP». Des del punt de vista de les matemàtiques,

Aquest article sorgeix d'una conferència pronunciada per l'autora a la Jornada P *versus* NP a la Facultat de Matemàtiques de la Universitat de Barcelona el 24 de novembre de 2010.

però, és una de les preguntes més importants plantejades per a l'estudi de la computació.

La influència d'aquest problema en la relació entre la ciència de la computació i les matemàtiques té dues vessants. D'una banda, per descomptat, és una font de motivació per al desenvolupament de tècniques matemàtiques innovadores. De l'altra, la seva dificultat aparent podria haver donat lloc a una tendència general a evitar un enfocament matemàtic en les àrees més pràctiques de la informàtica. De fet, en algunes àrees d'investigació, és habitual presentar una prova de NP-completesa per a motivar l'abandonament dels mètodes matemàtics i justificar l'estudi d'heurístiques i experiments.

Per començar amb una interpretació del problema «P *versus* NP» màximament divulgativa però tanmateix correcta, imaginem-nos la situació següent: tenim la tasca de resoldre qualsevol sudoku escollit per un adversari, que podrà ser de qualsevol mida (4×4, 9×9, 16×16, etc.), i a més l'hem de resoldre en temps fitat per una funció polinòmica, escollida per nosaltres anteriorment, de la mida del sudoku. (Per exemple, en 4^4 segons si és de mida 4×4, en 9^4 segons si és de mida 9×9, en 16^4 segons si és de mida 16×16, etc.)

La conjectura que $\mathrm{P} \neq \mathrm{NP}$ és equivalent a l'afirmació que aquesta tasca ens resultarà impossible si fem servir un mecanisme de còmput estàndard com per exemple la màquina de Turing —un model matemàtic de màquina que resulta ser tan potent com qualsevol ordinador digital amb memòria arbitràriament gran.[1]

La màquina de Turing és un model matemàtic d'ordinador que fa que el problema «P *versus* NP», o bé, equivalentment, el problema de determinar si efectivament $\mathrm{P} \neq \mathrm{NP}$, estigui definit molt precisament i sense lloc per a diferents interpretacions. P i NP són dues classes de problemes definides mitjançant aquesta màquina, i la qüestió és si són la mateixa classe o no. En la secció 3 presentarem la definició de la *màquina de Turing* i les definicions de P i NP. En la secció 4 veurem diverses maneres de reinterpretar el problema. En la secció 5 presentarem una conjectura sobre circuits, la demostració de la qual implicaria la resolució del problema «P *versus* NP», a més d'un teorema de Razborov de l'any 1985 que resol un cas especial d'aquesta conjectura. La darrera secció està dedicada al que es podria interpretar com la primera formulació per escrit del problema «P *versus* NP» en una carta de Gödel a Von Neumann de l'any 1956 que es va descobrir l'any 1988.

Abans d'entrar en els detalls tècnics de les definicions de P i NP, mirarem alguns exemples de problemes computacionals per a fer-nos una idea més informal i atractiva del concepte d'*algoritme*.

1 Cal distingir la situació de tenir un ordinador quàntic funcionant o una font d'aleatorietat perfecta, que encara no sabem si són models equivalents a la màquina de Turing clàssica quan es té en compte el temps de computació, però tampoc tenim cap raó per a creure que això canviaria la situació. En qualsevol cas, per ara, aquests models de màquina de computació no són físicament realitzables.

2 Algoritmes

Un *algoritme* és una recepta per a resoldre un problema de manera sistemàtica. És important distingir entre el concepte de *problema* i el concepte de *instància* d'aquest problema. Per exemple, un sudoku concret com els que ens trobem als diaris, determinat pels nombres que apareixen en algunes de les seves cel·les, és una instància del problema computacional anomenat SUDOKU. Farem servir aquesta tipografia sempre per a denotar problemes computacionals. Un algoritme que resolgui un problema computacional ha de ser prou general per a poder resoldre qualsevol instància donada del problema.

2.1 Camí eulerià

L'any 1735, Euler va resoldre el trencaclosques dels ponts de Königsberg: per la ciutat de Königsberg (avui Kaliningrad, Rússia) hi passa un riu on hi ha dues illes i set ponts que connecten les quatre parts de la ciutat. Mireu la figura 1. La pregunta del trencaclosques és si existeix alguna ruta que comenci en alguna de les quatre parts i passi per cada pont exactament una vegada. L'anàlisi que Euler va fer d'aquest problema es considera el principi de la teoria de grafs. En realitat, el cas dels ponts de Königsberg és una instància d'un problema computacional més general que és el que va analitzar Euler: donat un graf format per vèrtexs (les parts de terra) i arestes (els ponts que connecten dues parts), volem saber si existeix algun camí que passi per cada aresta exactament una vegada. Avui dia d'un tal camí en diem *Camí eulerià*. El problema d'esbrinar l'existència de camins eulerians també l'anomenarem CAMÍ EULERIÀ.

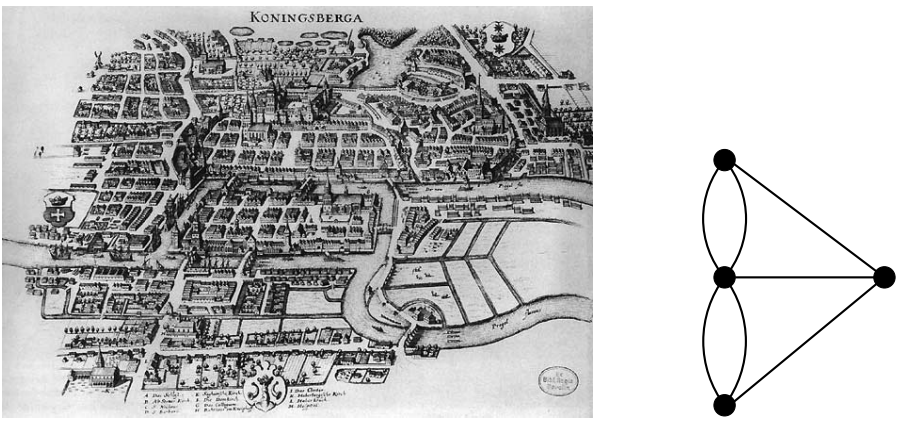

FIGURA 1: El mapa de Königsberg i el graf corresponent. Els punts són els vèrtexs del graf que corresponen a les parts de terra i les connexions són les arestes que corresponen als ponts.

Una manera de resoldre qualsevol instància d'aquest problema és simplement provant totes les possibilitats. Qualsevol vèrtex podria ser el punt de partida d'un camí eulerià i els hem de provar tots. La primera aresta podria ser qualsevol de les que surten del primer vèrtex i les hem de provar totes. Per recordar quines hem provat, podem definir un ordre de les arestes incidents a cada vèrtex i les anem provant en aquest ordre. Quan arribem al vèrtex següent tornem a tenir diverses opcions —totes les arestes que en surten que encara no estiguin travessades en el camí actual. Les provem totes en ordre i repetim aquest procés fins que no hi hagi cap aresta no travessada per on continuar. Si existeix algun camí eulerià, tard o d'hora, el trobarem, i, si no, esgotarem totes les possibilitats i sabrem del cert que no n'hi ha cap. D'un algoritme com aquest, que prova totes les possibilitats, se'n diu *de cerca exhaustiva*.

Fent precisament això (amb un programa) es pot comprovar que a la ciutat de Königsberg hi ha exactament cent camins que no es poden estendre, i el més llarg passa per sis ponts.

En comptes de provar totes les possibilitats i arriscar-se a cometre errors, Euler va generalitzar el problema i va donar una caracterització dels grafs que tenen un camí eulerià i alhora un algoritme per a trobar-lo.

TEOREMA 1 (EULER). *Un graf té un camí eulerià si i només si té 0 o 2 vèrtexs amb un nombre senar d'arestes incidents (*i. e.*, vèrtexs de* grau *senar) i és connex (*i. e.*, conté com a mínim un camí entre cada dos vèrtexs).*

La prova que la condició és necessària serà senzilla un cop hàgim notat que cada vegada que un camí entra en un vèrtex, n'ha de sortir, tret del cas que sigui el primer o l'últim del camí. La prova que la condició també és suficient és un algoritme per a trobar el camí que descrivim a continuació.

Primer, si hi ha exactament dos vèrtexs amb un nombre senar d'arestes incidents, afegim una aresta que connecti aquests dos vèrtexs. D'aquesta manera hem reduït el problema a trobar un *circuit* eulerià, *i. e.*, un camí que comenci i acabi al mateix vèrtex. Comencem a qualsevol vèrtex v i construïm un camí, aresta per aresta, arbitràriament, escollint cada vegada una aresta que encara no ha estat recorreguda, fins que arribem a un vèrtex on no podem continuar perquè no té cap aresta incident que no hàgim recorregut abans. A causa de la condició sobre la paritat dels graus, aquest vèrtex només pot ser v. Traiem les arestes ja recorregudes i repetim el mateix procés començant des de qualsevol vèrtex ja visitat que tingui arestes incidents encara per recórrer. El circuit que trobem es pot unir amb el primer de la mateixa manera que els dos cercles d'un 8 es poden escriure amb un sol traç. Repetim el procés fins que no quedin més arestes per recórrer.

La caracterització d'Euler ens dóna un algoritme molt eficient per a comprovar si un camí eulerià existeix o no. Només cal mirar els graus de tots els vèrtexs. El temps que tarda (*i. e.*, el nombre de passos que fa) aquest algoritme depèn de com està representat el graf però en tot cas fa un nombre de passos que és proporcional a la mida de la representació del graf. Diem que aquest algoritme té *complexitat lineal*. En canvi, l'algoritme que prova tots els camins,

segons quin graf hagi de tractar, pot haver de mirar un nombre de camins que és *exponencial* en el nombre de vèrtexs.

En el cas de Königsberg la diferència entre lineal i exponencial potser no és gaire important. Però un cop vulguem resoldre instàncies més grans, com per exemple Venècia, que té quatre-cents nou ponts i cent disset illes, veurem clarament que l'algoritme exponencial no és sostenible.

2.2 Arbre generador mínim

Un *arbre generador* d'un graf és qualsevol subgraf connex acíclic que conté tots els vèrtexs del graf. Mireu la figura 2. En el cas d'un graf amb pesos positius, el problema que ens interessa és el de trobar l'arbre generador amb pes mínim. L'algoritme per a trobar un tal arbre resulta sorprenentment senzill i eficient i és un exemple d'*algoritme voraç*: escollim les arestes de l'arbre en ordre de pes. Cada cop escollim l'aresta de pes mínim d'entre les que no generen cap cicle. Quan no es pugui afegir cap aresta ja tenim un arbre generador i, per increïble que sembli, de pes mínim. Aquest és un algoritme clàssic descobert per Kruskal l'any 1956 [13].

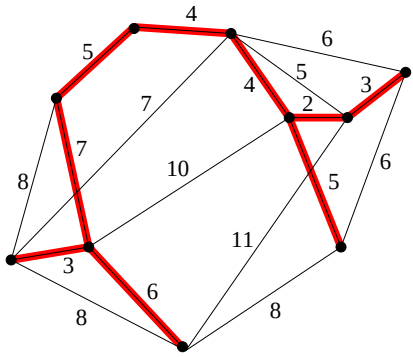

FIGURA 2: Arbre generador de pes mínim.

Abans de demostrar que l'arbre resultant és de pes mínim, mirem-nos un moment la complexitat d'aquest algoritme, *i. e.*, el màxim nombre de passos o operacions que fa sobre un graf qualsevol de n vèrtexs i m arestes. El nombre exacte depèn de la representació del graf i de la definició d'operació bàsica. Per això, en algorítmia mai parlem del nombre exacte d'operacions; en canvi, parlem de l'ordre de magnitud del nombre d'operacions (depenent de quin tipus d'ordre estem parlant —fites inferiors, superiors, estrictes o no— fem servir la notació asimptòtica Ω, ω, O, o, Θ). Per exemple, fer set operacions per a cada parell de vèrtexs es considera del mateix ordre de magnitud que fer una operació per a cada parell de vèrtexs. En ambdós casos diem que fem un nombre d'operacions com a molt quadràtic en el nombre de vèrtexs, o $O(n^2)$. Una altra manera de dir el mateix és que l'algoritme fa com a molt $c \times n^2$ passos, per a una constant c que no depèn de n.

Considerem una implementació senzilla que no fa servir cap estructura de dades sofisticada. Associem a cada component connexa del subgraf construït un nombre entre 1 i n i per a cada vèrtex guardem una etiqueta que digui a quina component pertany. Al principi l'etiqueta és la mateixa que el nombre del vèrtex. Cada cop que afegim una aresta al subgraf, connectant un vèrtex amb etiqueta i amb un vèrtex amb etiqueta j, canviem totes les etiquetes j per i (la component etiquetada j desapareix). Per trobar l'aresta de pes mínim d'entre les que no generen cicles, primer mirem totes les arestes una per una i guardem l'aresta de pes mínim que connecta dos vèrtexs amb etiquetes diferents, i segon, canviem les etiquetes dels vèrtexs de les components que aquesta aresta connecta. Per tant, l'algoritme fa $O(m+n)$ operacions per a cada aresta que afegeix a l'arbre. En total són $O(n(m+n))$ passos, que per a un graf connex són $O(nm)$ passos. Amb una estructura de dades no trivial la complexitat de l'algoritme es pot reduir a $O(m \log m)$ operacions.

Demostrarem que l'arbre generat per l'algoritme voraç té pes mínim. Suposem que l'algoritme fa algun error i sigui $e = (u, v)$ la primera aresta errònia. Sigui $A \subset E$ el conjunt d'arestes escollides abans que e, on E és el conjunt de totes les arestes del graf. Com que e és el primer error, existeix un arbre generador mínim que conté A però no e; sigui $T \supset A$ aquest arbre amb $e \notin T$. A més, no existeix cap arbre generador del mateix pes que contingui $A \cup \{e\}$ (altrament e no seria estrictament el primer error).

Segons l'algoritme, qualsevol aresta $e' \in E \backslash A$ de pes menor que el pes de e genera un cicle junt amb A i, per tant, no pertany a T. D'altra banda, e genera un cicle si l'afegim a T i aquest cicle ha de contenir una aresta que no pertany a A, perquè $A \cup \{e\}$ és acíclic. Sigui e' aquesta aresta. Com que no pertany a A, ha de tenir pes com a mínim igual al pes de e. Llavors $T \backslash \{e'\} \cup \{e\}$, el qual és un arbre generador, té pes igual o menys que el pes de T i conté $A \cup \{e\}$. Això contradiu la definició de e.

2.3 Circuit hamiltonià i el problema del viatjant

La història del problema Circuit Hamiltonià s'assembla molt a la del problema Camí Eulerià. L'any 1857, mentre estava treballant en una àlgebra que va anomenar *icosiana* perquè estava basada en les simetries de l'icosàedre, Hamilton es va plantejar el problema de determinar si existeix algun camí tancat, *i. e.*, un *circuit*, per les arestes del dodecàedre que passi per cada vèrtex exactament una vegada. La solució es pot veure en la figura 3. Malauradament, el seu mètode, que té a veure amb l'àlgebra d'icosians, no es pot generalitzar a qualsevol graf.

A diferència de Camí Eulerià, fins avui encara no tenim cap algoritme eficient per a trobar circuits hamiltonians en grafs arbitraris. L'algoritme de cerca exhaustiva tarda un temps proporcional al nombre de camins en el graf, que pot ser proporcional a $n!$ per a un graf de n vèrtexs. Existeix un algoritme millor, que fa $O(2^n n^2)$ passos [11]. Durant gairebé cinquanta anys aquest algoritme de Held i Karp era el millor que es coneixia. L'any passat Björklund

va fer un gran avenç i va presentar un algoritme exponencialment més ràpid que fa $O(1{,}657^n)$ passos [4].

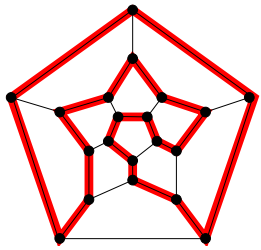

FIGURA 3: Circuit hamiltonià pel dodecàedre.

Vegem l'algoritme clàssic de Held-Karp de l'any 1962, que és un dels primers exemples de la tècnica anomenada *programació dinàmica*. Sigui V el conjunt de vèrtexs del graf, sigui v un vèrtex arbitrari i $|V| = n$. Per a cada vèrtex $u \neq v$ del graf i per a cada subconjunt de vèrtexs $X \subseteq V\backslash\{u, v\}$ determinarem si existeix algun camí que comenci a v, acabi a u i passi per tots els vèrtexs de X exactament una vegada. Sigui $T(u, X) = 1$ si existeix tal camí i $T(u, X) = 0$ si no. Calcularem $T(u, X)$ per a tot X de mida 0, 1, 2, etc. en aquest ordre. Primer per X amb $|X| = 0$ tenim $T(u, \emptyset) = 1$ si i només si hi ha una aresta entre u i v. Per a cada $X \subseteq V\backslash\{u, v\}$ podem calcular $T(u, X)$ fent servir els valors de $T(w, X\backslash\{w\})$ per $w \in X$. Efectivament, $T(u, X) = 1$ si i només si existeix algun vèrtex $w \in X$ amb una aresta entre w i u i $T(w, X\backslash\{w\}) = 1$. Finalment, si existeix algun u tal que $T(u, V\backslash\{u, v\}) = 1$ i amb una aresta entre u i v, llavors podem dir que existeix un circuit hamiltonià, i viceversa.

Per a calcular $T(u, X)$, necessitem comprovar només $|X| < n$ possibilitats. S'han de calcular $n2^n$ valors diferents de $T(u, X)$; per tant, aquest algoritme fa $O(n^2 2^n)$ passos.

Una extensió del problema del CIRCUIT HAMILTONIÀ és el PROBLEMA DEL VIATJANT, on les arestes tenen pes (com podria ser per exemple la distància entre dues ciutats) i l'objectiu és trobar un circuit de pes total mínim que passi per tots els vèrtexs com a mínim una vegada. Aquest problema també es pot resoldre amb l'algoritme de Held-Karp, canviant $T(u, X)$ perquè indiqui el camí de pes mínim d'entre els que comencen a v, acaben a u i passen per tots els vèrtexs de X com a mínim una vegada (si n'hi ha, i, si no, és igual a ∞).

Si no sabem trobar el circuit de pes mínim, un objectiu alternatiu que ens podem plantejar és trobar un circuit el pes del qual sigui com a molt un factor finit més gran que l'òptim. Aquest és l'objectiu de l'àrea d'algoritmes d'aproximació. En el cas del PROBLEMA DEL VIATJANT existeix un algoritme eficient per a trobar un circuit que té pes com a molt dues vegades més que l'òptim. Per a trobar-lo, aprofitem l'algoritme de l'arbre generador mínim. L'arbre generador ens dóna una manera de visitar tots els vèrtexs, i el circuit, que repeteix cada aresta de l'arbre dues vegades, té pes dues vegades més

que el pes de l'arbre. Com sabem que aquest no és molt pitjor que l'òptim? Perquè el circuit òptim també es pot convertir en un arbre generador (traient-li les arestes que generen cicles) i, per tant, té pes més gran que el de l'arbre generador mínim.

En el cas que els vèrtexs són punts en $\mathbb{R}^n$ i les distàncies són euclidianes, existeix un algoritme d'Arora i Mitchell de 1998 [3, 15] que, donat qualsevol $\epsilon > 0$, troba un circuit de pes com a molt $1 + \epsilon$ vegades més gran que l'òptim i té complexitat polinòmica en $1/\epsilon$ i la mida del graf.

2.4 Clica

Una *clica* (*clique* en anglès) d'un graf és un conjunt de vèrtexs completament connectats. El problema CLICA és el de determinar si existeix una clica d'una mida donada en un graf. Si busquem una clica de mida k la cerca exhaustiva prova cada subconjunt de vèrtexs de mida k i, per tant, fa uns n^k passos. No es coneix cap algoritme que faci un nombre de passos polinòmic en n i k.

Igual que en el cas del CIRCUIT HAMILTONIÀ, es coneix un algoritme exponencial que és millor que la cerca exhaustiva [16]. L'algoritme té complexitat $O(n^{kc/3})$, on c és una constant que ve de l'exponent de la complexitat de l'algoritme més eficient per a multiplicar matrius, que avui dia és $O(n^{2,376})$ [7]. Observeu que l'algoritme que aprenem a classe per a multiplicar matrius fa $O(n^3)$ passos.

3 Definicions de P i NP

3.1 Computació, computabilitat i autòmats

Tots els exemples de la secció anterior es poden resoldre simplement comprovant totes les possibilitats, que són, en tots els casos, un nombre finit. Tanmateix, existeixen molts problemes per als quals no hi ha cap algoritme per a saber si el problema té solució. Per definir matemàticament el concepte de *computabilitat*, Turing va definir un model de màquina que avui es coneix com la *màquina de Turing* [22]. Aquesta definició de *computabilitat* va resultar ser equivalent a altres definicions anteriors que eren significativament diferents. De fet la tesi de Church-Turing formulada en aquella època, i que fins avui dia encara no ha estat refutada, postula que no existeix cap model de computació físicament implementable que pugui resoldre problemes computacionals no resolubles per màquines de Turing.[2]

En aquesta secció ens centrem en el cas particular dels *problemes de decisió*, els que tenen resposta «sí» o «no». Tots els problemes de la secció anterior es poden reduir fàcilment a problemes de decisió. Per exemple, podem trobar l'arbre generador de mínim cost, preguntant per a cada aresta si aquesta està o no inclosa en un arbre generador mínim i, en el cas que hi estigui, contraient-la i

2 Això continua sent cert fins i tot si assumim que els ordinadors quàntics són físicament implementables, perquè la tesi de Church-Turing no té en compte el temps de computació.

repetint el procés amb el multigraf resultant. Serà convenient, doncs, identificar un problema de decisió amb el conjunt de les instàncies positives —les que tenen resposta «sí». D'aquest conjunt se'n diu un *llenguatge*; és un subconjunt de les cadenes de lletres d'un alfabet predeterminat. Per exemple, podem definir el llenguatge «graf eulerià» que conté (les codificacions en 0 i 1 de) tots els grafs eulerians.

Es poden definir diversos models matemàtics de màquines per a reconèixer llenguatges. Vegem-ne un exemple elemental.

Suposem que el llenguatge que ens interessa consisteix en totes les paraules escrites en l'alfabet $\Sigma = \{a, b\}$ que tenen un nombre de a que és divisible per 3 o 4. Hi ha un tipus de màquina senzill que serveix per a aquest llenguatge, que es coneix com *autòmat finit determinista* (AFD). Un AFD consta de quatre components: un conjunt d'*estats* Q, un subconjunt dels estats que són els estats d'*acceptació* $A \subseteq Q$, un estat d'*inici* $q^0 \in Q$, i una funció de *transició* $\delta\colon Q \times \Sigma \to Q$. Diem que l'autòmat accepta una paraula $(a_1, \ldots, a_n) \in \Sigma^n$ si existeix una seqüència d'estats $q_0, q_1, \ldots, q_n \in Q$, tal que $q_0 = q^0$, $q_i = \delta(q_{i-1}, a_i)$ per a tot $i \in \{1, \ldots, n\}$ i $q_n \in A$. Això simplement vol dir que l'autòmat llegeix la paraula lletra a lletra i a cada pas canvia el seu estat depenent de l'estat actual i de la lletra que ha llegit. Si acaba en un estat d'acceptació, llavors diem que l'autòmat accepta la paraula. El conjunt de paraules acceptades és el llenguatge reconegut per aquest autòmat. L'autòmat il·lustrat en la figura 4 reconeix el llenguatge de paraules de l'alfabet $\{a, b\}$ que contenen un nombre de a divisible per 3 o 4.

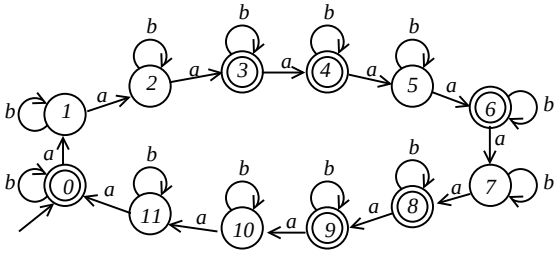

FIGURA 4: AFD que accepta les paraules que tenen un nombre de símbols a que és divisible per 3 o 4. Els estats d'aquesta màquina $Q = \{0, \ldots, 11\}$ indiquen quants a ha llegit, mòdul 12. L'estat d'inici és 0.

La màquina de Turing és només una mica més sofisticada que l'AFD en el fet que té *memòria*. La *memòria* és una cinta potencialment infinita dividida en posicions on es poden guardar lletres —una lletra per posició— i on al principi hi ha escrita l'*entrada* —la paraula que s'ha de decidir si pertany al llenguatge o no. Hi ha un capçal col·locat al principi de la cinta que es pot moure una posició en les dues direccions o quedar-se al mateix lloc, i mitjançant aquest capçal es poden llegir i escriure lletres a la cinta. Per acomodar-hi la cinta, canviem la funció de transició. Aquesta funció especifica tres coses, depenent de l'estat en què està la màquina i la lletra escrita sota el capçal: 1) quina lletra escrivim en el lloc del capçal; 2) cap on s'ha de moure el capçal —a l'esquerra, a la dreta o

cap moviment, que denotem respectivament amb -1, $+1$ i 0; 3) a quin estat ha d'estar la màquina després de la transició. Per tant, $\delta\colon Q\times\Sigma \to \Sigma\times\{-1,0,1\}\times Q$. Com que en aquest cas no és clar quan la màquina s'hauria de parar, definim dos estats especials $A, R \in Q$ —respectivament d'acceptació i de rebuig— de manera que si la màquina entra en algun d'aquests estats, para. Si para a l'estat d'acceptació, llavors diem que la paraula està en el llenguatge de la màquina, i, si para a l'estat de rebuig, llavors no hi és. El *llenguatge de la màquina* és el conjunt de les paraules en què la màquina para a l'estat d'acceptació. Per exemple, la figura 5 és una màquina de Turing per a reconèixer palíndroms.

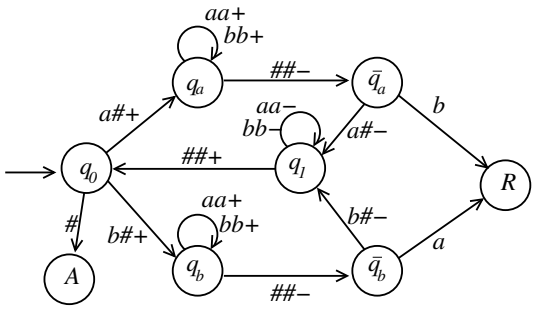

FIGURA 5: Màquina de Turing per a reconèixer palíndroms. Les etiquetes de tres símbols que porten les transicions són: la lletra que llegeix, la lletra que escriu, i la direcció en què es mou el capçal (fent servir les abreviacions − i + respectivament per a moure a l'esquerra i a la dreta). El símbol # denota una posició buida. Depenent de quina és la primera lletra, la màquina va a l'estat q_a o q_b, després el capçal es mou a la dreta fins al final de la paraula i quan arriba al final l'estat canvia a $\overline{q_a}$ o $\overline{q_b}$ respectivament. Si l'última lletra és diferent de la primera, la màquina entra a l'estat de rebuig R i, si no, torna al principi de la paraula, canvia l'estat a l'estat inicial q_0 i repeteix. Cada cop que llegeix la primera i l'última lletra les esborra i així la paraula decreix fins que la cinta queda buida. Llavors la màquina entra a l'estat d'acceptació A.

Turing també va introduir una màquina concreta, la *màquina universal*, que llegeix de la cinta la descripció d'una màquina de Turing qualsevol i una entrada, i simula el funcionament d'aquesta màquina per a aquesta entrada. Aquesta és, de fet, la base dels ordinadors actuals i és gràcies a això que no cal construir un ordinador diferent per a cada problema que vulguem resoldre.

Quan Turing va inventar la seva màquina encara no s'estudiava el *temps de computació*, *i. e.*, el nombre de passos que faria la màquina. Tampoc s'estudiava l'*espai de computació*, *i. e.*, la quantitat de cinta que faria servir, o la memòria. L'interès en aquestes qüestions va venir més tard, als anys 1950, quan es van començar a utilitzar màquines de computació a la pràctica. Llavors va ser necessari optimitzar la utilització dels dos recursos més importants —el temps i l'espai.

3.2 Determinisme *versus* indeterminisme

Hi ha una manera una mica tramposa, en el sentit que no és físicament realitzable, de millorar els dos models d'autòmat que hem descrit. Imagineu-vos que existissin mons paral·lels i que a cada pas la màquina pogués anar a parar no només a un estat, sinó a qualsevol d'un conjunt d'estats determinat per la funció de transició —cadascun en un món paral·lel diferent. Així el nombre de mons creix exponencialment amb el nombre de passos.[3] En aquest model, direm que l'autòmat accepta una paraula si ho fa en algun dels mons paral·lels. L'AFD s'ha convertit en un AFI (autòmat finit indeterminista) i la seva funció de transició ara és $\delta\colon Q \times \Sigma \to \mathcal{P}(Q)$. Per exemple, el llenguatge de les paraules que contenen un nombre de a divisible per 3 o 4 es pot reconèixer amb l'AFI de la figura 6. (En aquest exemple n'hi ha prou amb dos mons paral·lels, perquè la màquina té dues opcions només al primer pas, però en general podria tenir diverses opcions a cada pas.)

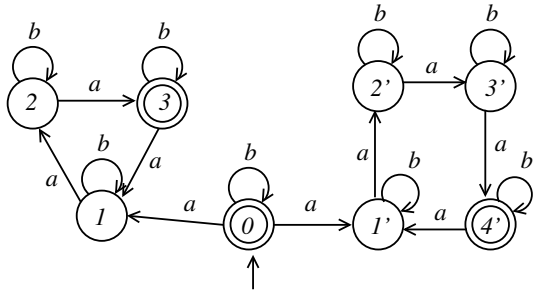

FIGURA 6: AFI que accepta les paraules que tenen un nombre de símbols a que és divisible per 3 o 4. Quan llegeix el primer a la màquina té l'opció d'anar cap a l'estat 1 o 1′.

El concepte d'*indeterminisme* s'estén també a la màquina de Turing. Ara podem definir les classes P i NP en una frase. Les dues són conjunts de llenguatges. (N)P conté tots els llenguatges que es poden reconèixer amb una màquina (in)determinista de Turing en temps polinòmic en la longitud de l'entrada. Amb això volem dir que el temps està fitat per una funció polinòmica de la mida de l'entrada. Com que una màquina determinista és un cas particular de màquina indeterminista, en què a cada pas només hi ha una opció, és clar que $\mathrm{P} \subseteq \mathrm{NP}$.

Explicat així, és obvi que les màquines de Turing indeterministes, amb aquest poder màgic per a calcular en un nombre exponencial de mons paral·lels, són més potents que les màquines deterministes, en el sentit que permeten calcular molt més ràpid. Però fins a quin punt ens ajuda l'indeterminisme? De fet, podria ser que tot el que podem calcular en temps polinòmic amb la màquina indeterminista, també ho poguéssim calcular en temps polinòmic amb

3 És precisament a causa d'aquest creixement exponencial que el model no seria físicament realitzable encara que disposéssim d'una màquina amb molts processadors treballant en paral·lel.

una màquina determinista (i llavors P = NP), encara que aquest polinomi fos de grau molt més alt.

3.3 Certificats

Hi ha una altra manera de pensar en la classe de problemes NP que no té res a veure amb l'indeterminisme. Imagineu-vos que cada paraula d'un llenguatge hagués de presentar un certificat de pertinença al llenguatge. Aquest certificat no hauria de ser gaire llarg —polinòmic en la longitud de la paraula— i s'hauria de poder verificar eficientment —en temps polinòmic— fent servir una màquina de Turing determinista. Considerem la classe dels llenguatges per als quals existeixen certificats polinòmics per a totes les paraules del llenguatge i a més existeix una màquina de Turing determinista per a verificar aquests certificats en temps polinòmic en la longitud del certificat (i de la paraula). Vegem-ne un exemple: el llenguatge «graf hamiltonià». Cada graf hamiltonià pot presentar un circuit hamiltonià com un certificat de pertinença al conjunt de grafs hamiltonians. La màquina que verifica aquests certificats simplement comprova (en temps lineal) que el circuit presentat realment és hamiltonià.

Resulta que el conjunt d'aquests llenguatges és exactament NP. En una direcció, la màquina de Turing indeterminista ens permet definir un certificat: per a cada paraula del llenguatge existeix com a mínim un món de tots els mons paral·lels on la màquina de Turing indeterminista ha arribat a un estat d'acceptació. La descripció d'aquest món —la seqüència de passos que la màquina ha d'escollir per a arribar a l'estat d'acceptació— és un certificat de longitud polinòmica. La màquina que comprova això (en temps lineal en la mida del certificat) simplement comprova que tots els passos són legals. En l'altra direcció, podem construir una màquina de Turing indeterminista que simplement genera totes les cadenes de lletres (al mateix temps en els diversos mons paral·lels) i en cada món comprova si aquesta cadena és un certificat (simulant la màquina que verifica certificats). Si és així, entra a l'estat d'acceptació, altrament, al de rebuig.

Ara no és difícil comprovar que les versions de decisió de tots els exemples de la secció anterior pertanyen a NP.

Aquesta interpretació de la conjectura a través de certificats té conseqüències filosòfiques tan o més interessants que la utilitat de tenir diversos mons paral·lels. Creure en la conjectura que P no és el mateix que NP és una manera de postular que existeix una diferència fonamental entre el procés creatiu de buscar solucions i el procés mecànic de verificar certificats.

3.4 Jerarquia temporal

De vegades es confon NP amb el conjunt de problemes per als quals existeix un algoritme que triga un temps exponencial (en màquines de Turing deterministes). De fet, d'aquesta classe de problemes se'n diu EXP i és clar que NP $\subseteq$ EXP, però la pregunta sobre si el recíproc també és cert és un altre problema

obert. Una cosa que podem assegurar és que P $\subsetneq$ EXP. Aquesta afirmació és conseqüència del teorema de la jerarquia temporal, que estableix que per a qualsevol funció $f(n)$ existeixen problemes resolubles en temps $f(n)^2$ que no ho són en temps $f(n)$. La demostració d'aquesta afirmació utilitza la tècnica de la diagonalització (la mateixa tècnica que va inventar Cantor per demostrar que els nombres reals són més, en cardinalitat, que els enters).

4 NP-completesa

Per a demostrar que P no és igual a NP és suficient demostrar, per a només un problema a NP, que no existeix cap algoritme polinòmic que el resolgui. D'altra banda, per a demostrar que P és igual a NP sembla que tenim molta feina. Sembla que hauríem de continuar indefinidament inventant algoritmes per a més i més problemes. Però de fet no cal. L'any 1971 Cook a Nord-amèrica [6] i Levin a la Unió Soviètica [14] (independentment) van demostrar que n'hi havia prou amb un algoritme polinòmic, per a un problema determinat conegut com a SAT, i aquest algoritme serviria per a resoldre tots els problemes a NP.

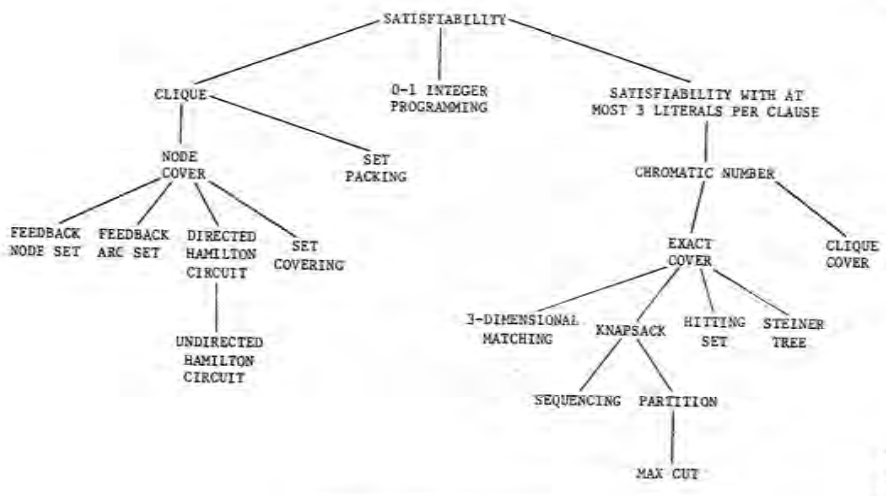

FIGURA 7: La figura de l'article original de Karp [12] mostra la seqüència de reduccions per a demostrar que vint problemes estàndard també són NP-complets.

Un any més tard Karp va demostrar que aquest problema no és l'únic amb aquesta propietat [12]. Karp va identificar vint problemes que s'havien resistit als intents dels matemàtics per a trobar-hi solucions eficients, i va demostrar que tots eren tan difícils com SAT. Ho va fer mitjançant reduccions de SAT a cadascun d'aquests problemes en una seqüència de reduccions que es pot veure en la figura 7. De tots els problemes que tenen aquesta propietat se'n

diuen NP-*difícils* perquè són almenys tan difícils de resoldre com qualsevol problema a NP. Si un problema és NP-difícil i a més és de NP, també se'n diu NP-*complet*.

Vegem-ne un exemple. Presentarem una reducció des del problema PARTICIÓ al problema MAX-CUT (tall màxim) [12]. El problema PARTICIÓ és: donada una col·lecció de nombres, determinar si hi ha una subcol·lecció la suma de la qual sigui la meitat de la suma total. Per exemple, per a la col·lecció $(1, 5, 3, 9, 6, 6, 10)$ la subcol·lecció $(3, 5, 6, 6)$ té suma la meitat del total. El problema MAX-CUT és: donat un graf amb pesos a les arestes i un nombre K, determinar si hi ha una partició dels vèrtexs en dues parts de manera que la suma dels pesos de les arestes que connecten vèrtexs de les dues parts diferents sigui almenys K. La figura 8 és un exemple de tall de pes 31. Una reducció de PARTICIÓ a MAX-CUT és un algoritme (polinòmic) per convertir qualsevol instància A de PARTICIÓ en una instància B de MAX-CUT tal que A és una instància «sí» si i només si B és una instància «sí». Donada la seqüència $(a_1, a_2, \ldots, a_n)$, amb $A = \sum_{i=1}^{n} a_i$, definim $K = A^2/4$ i sigui G un graf complet de n vèrtexs $\{1, 2, \ldots, n\}$. L'aresta que connecta el vèrtex i amb el vèrtex j té pes $a_i \times a_j$. Això defineix la reducció. Fixeu-vos que el pes total d'un tall que separa un subconjunt dels vèrtexs $S \subseteq \{1, 2, \ldots, n\}$ és $\sum_{i \in S} a_i \times \sum_{i \notin S} a_i = (\sum_{i \in S} a_i)(A - \sum_{i \in S} a_i)$. Atès que $x(A - x) \leq A^2/4$ i el màxim s'assoleix quan $x = A/2$, podem concloure que hi ha un tall de pes almenys K si i només si hi ha una partició dels nombres en dues parts que sumen igual.

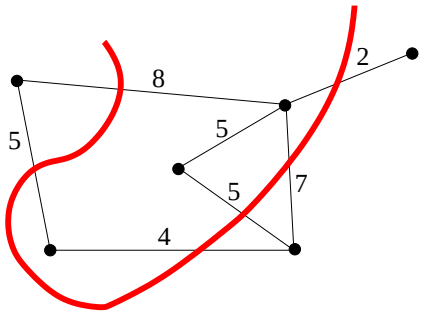

FIGURA 8: El tall màxim en aquest cas talla totes les arestes menys una i té pes 31.

Hi ha problemes que són NP-difícils en el cas general però les instàncies d'un conjunt més restringit són resolubles en temps polinòmic. Per exemple, el problema de coloració de grafs amb quatre colors en general és NP-difícil, però si ens centrem en els grafs planars, llavors tots són 4-colorables pel teorema dels quatre colors i, per tant, el problema resulta trivial. I la coloració de grafs amb tres-colors? En general, el problema també és NP-difícil. Però en aquest cas, es pot demostrar que, fins i tot si restringim el problema a grafs planars, aquest continua sent NP-difícil. A continuació, presentem una reducció des del problema de la 3-coloració de grafs generals a la 3-coloració de grafs planars [9].

Suposem que el graf està representat al pla possiblement amb encreuaments de dues arestes (si en algun punt se'n creuen tres o més, pertorbem una mica la representació). Canviem els encreuaments que travessen una aresta per un graf planar com mostra la figura 9. Cada rombe amb W dins denota una còpia del graf W de l'esquerra. El graf W té la propietat que, en qualsevol coloració vàlida de W amb tres colors, els vèrtexs x i x' tenen el mateix color, igual que y i y', i tanmateix els colors de x i y no estan restringits de cap manera. D'aquest tipus de construcció se'n diu *giny* (*gadget* en anglès) i és una tècnica que es fa servir molt sovint per a reduir un problema a un altre. El graf que obtenim substituint tots els encreuaments per W és un graf planar que es pot colorar amb tres colors si i només si el graf original es pot colorar amb tres colors.

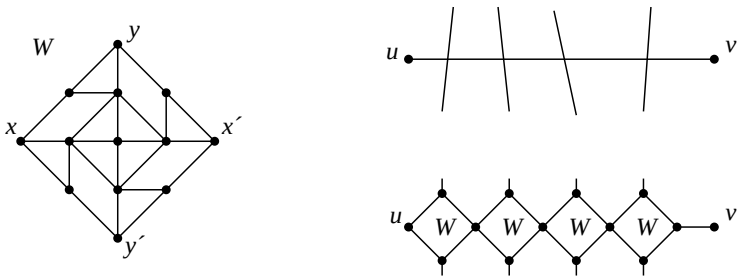

FIGURA 9: El giny per a reduir el problema de coloració de grafs amb tres colors al problema de coloració de grafs *planars* amb tres colors. Reemplacem tots els encreuaments d'arestes al pla pel graf planar W.

Al principi d'aquest article hem donat l'exemple del problema SUDOKU. El cas és que l'any 2002 Takayuki Yato a la seva tesi de màster va demostrar que aquest problema és NP-complet mitjançant una reducció des del problema QUADRATS LLATINS [5]. L'única motivació per a estudiar la complexitat de SUDOKU és la popularitat del joc 9×9. Però aquest és només un de molts problemes naturals que han sorgit durant els anys. Avui dia l'arbre de reduccions iniciat per Karp conté milers de problemes de diversos camps —la biologia, l'enginyeria, l'economia, etc.

5 Complexitat de circuits

No sorprendrà ningú el fet que les màquines de Turing tenen alguna cosa a veure amb els circuits electrònics. I què són els circuits? No són més que un tipus de graf dirigit acíclic amb algunes etiquetes als vèrtexs. Per exemple, un circuit amb portes lògiques AND, OR i NOT té vèrtexs amb aquestes tres etiquetes a més de vèrtexs d'entrada que només tenen arestes de sortida i un vèrtex de sortida que té només una aresta d'entrada. Un circuit executa un còmput de la manera següent: si posem valors booleans «cert» i «fals» als vèrtexs d'entrada, aquests determinen el valor dels altres vèrtexs aplicant les

funcions lògiques que els etiqueten, fins que s'arriba al vèrtex de sortida (vegeu la figura 10).

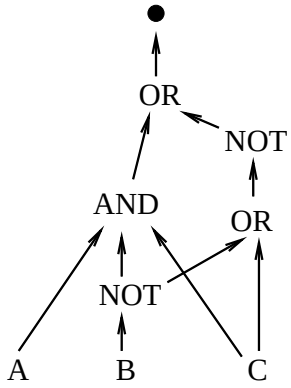

FIGURA 10: Un circuit que calcula el valor de la fórmula «(A i (no B) i C) o (no ((no B) o C))».

Potser podríem haver definit el concepte de *computació* fent servir circuits des del principi, doncs? En part sí, però el problema amb els circuits és que cada circuit opera només amb paraules d'entrada d'una mida predeterminada, perquè el nombre de portes d'entrada és fix. Tanmateix, ja als anys vuitanta, els investigadors van pensar que raonar sobre la mida de circuits podria ser més fàcil que raonar sobre el temps que tarda una màquina i d'aquí que pensessin en maneres de relacionar-los. L'estudi de la complexitat de circuits és encara ara una de les direccions més prometedores per a l'estudi de la conjectura «P *versus* NP».

La qüestió de si NP està inclosa en P té un anàleg al món dels circuits que és si NP està inclosa en P/poly. P/poly és també una classe de problemes. Són els problemes que es poden resoldre fent servir una família de circuits amb portes AND, OR i NOT, un circuit per a cada mida d'entrada, amb el nombre de portes polinòmic en la mida de l'entrada. De fet no és difícil convèncer-se un mateix que si un problema es pot resoldre amb una màquina de Turing en temps polinòmic, llavors també es pot resoldre amb una família de circuits de mida polinòmica. El recíproc també és cert però només si considerem famílies de circuits uniformes. Diem que una família de circuits $C_1, C_2, \ldots$ per a mides d'entrada $1, 2, \ldots$ respectivament és *uniforme* si existeix una màquina de Turing per a construir C_n en temps polinòmic en n. Amb aquesta definició podem afirmar que P és la classe de problemes resolubles fent servir famílies de circuits polinòmics uniformes, mentre que P/poly és la classe de problemes resolubles fent servir circuits polinòmics (no necessàriament uniformes). Una possibilitat, tot i que improbable, és que hi hagi famílies de circuits polinòmics per a tots els problemes de NP, però que no es puguin construir en temps polinòmic (*i. e.* que $\mathrm{NP} \subseteq \mathrm{P/poly}$ però $\mathrm{NP} \not\subseteq \mathrm{P}$). En tot cas, és clar que si es demostra que $\mathrm{NP} \not\subseteq \mathrm{P/poly}$ s'haurà establert que $\mathrm{P} \neq \mathrm{NP}$.

5.1 Circuits monòtons

Quan una conjectura es mostra difícil és natural començar amb casos especials. Una manera de fer-ho que va proposar Razborov l'any 1985 és concentrar-se primer en els circuits de portes AND i OR (deixar el NOT a banda) i demostrar que fent servir només aquestes portes i només poques vegades (un nombre polinòmic) hi ha algun problema de NP que no podem resoldre [18]. Per tal que aquesta fita tingui sentit, cal escollir un problema de NP que es pugui resoldre només amb les portes AND i OR. En particular, el problema havia de ser monòton. Razborov va escollir el problema CLICA de la secció 2.4. Aquest és un problema monòton, perquè si la resposta és «sí», quan afegim una aresta segueix sent «sí».

Sigui $\text{CLICA}_{n,k}$ la restricció del problema CLICA a grafs de n vèrtexs i cliques de mida k. L'entrada dels circuits per $\text{CLICA}_{n,k}$ és la matriu d'adjacència del graf, representable per $\binom{n}{2}$ variables $g_{i,j}$, una per a cada parell de vèrtexs diferents i i j que indiquen si hi ha una aresta entre i i j. Un exemple de circuit monòton per a resoldre $\text{CLICA}_{n,k}$ és simplement

$$\text{OR}_{S\subseteq V,|S|=k}\left(\text{AND}_{i\neq j\in S}\, g_{i,j}\right).$$

Aquest circuit té $\binom{n}{k}+1$ portes. Si considerem $k=\sqrt[4]{n}$, per exemple, això és un nombre exponencial de portes. Razborov va demostrar que no es pot fer gaire més bé que això:

TEOREMA 2 (RAZBOROV [18]). *Cada circuit monòton per a resoldre* $\text{CLICA}_{n,k}$ *amb* $k=\left\lceil\sqrt[4]{n}\right\rceil$ *té almenys* $n^{\sqrt[8]{n}/12}$ *portes.*

Aquí donarem un esbós de la demostració i per als detalls adrecem el lector al llibre de Papadimitriou [17]. Per a una exposició bastant més profunda recomanem al lector curiós una exposició de Gowers [10] que explica una manera més natural (o més fonamental si voleu) d'arribar a la demostració.

Abans de començar demostrem un lema combinatori que ens serà molt útil. Una *flor* de mida p és un conjunt de p subconjunts d'un univers tal que tots els parells de subconjunts tenen la mateixa intersecció —d'aquesta intersecció en diem el *cor* de la flor.

LEMA 3 (ERDÖS-RADO). *Sigui* $\mathcal{Z}$ *una família de més de* $M=(p-1)^{\ell}\ell!$ *conjunts diferents no buits d'un univers comú, tots de mida* ℓ *o menys. Llavors* $\mathcal{Z}$ *conté una flor de mida* p*.*

PROVA. La demostració és una inducció en ℓ. Per a $\ell=1$, una família de com a mínim p subconjunts de mida 1 conté p elements. Aquests formen una flor amb cor buit.

Suposem que $\ell\geq 1$. Sigui $\mathcal{D}$ un subconjunt maximal de $\mathcal{Z}$ de conjunts disjunts, *i. e.*, qualsevol altre conjunt de $\mathcal{Z}$ té intersecció no buida amb algun dels conjunts de $\mathcal{D}$. Si $\mathcal{D}$ té mida p o més llavors conté una flor amb cor buit. Si no la mida de la unió dels conjunts, $D=\cup_{S\in\mathcal{D}}S$, és $|D|\leq(p-1)\ell$ i, com que

cada conjunt de $\mathcal{Z}$ interseca D, llavors existeix algun element a D que apareix com a mínim en $\frac{M}{(p-1)\ell}$ conjunts de $\mathcal{Z}$. Sigui d aquest element. Considerem la família d'aquests conjunts, traient l'element comú: $\mathcal{Z}'_d = \{Z - \{d\} : Z \in \mathcal{Z}, d \in Z\}$. Tots aquests conjunts tenen mida com a màxim $\ell - 1$ i a més $|\mathcal{Z}'_d| \geq (p-1)^{\ell-1}(\ell-1)!$. Per tant, per la hipòtesi d'inducció, $\mathcal{Z}'_d$ conté alguna flor. Tornem a afegir l'element d als conjunts de la flor i obtenim una flor de $\mathcal{Z}$. □

PROVA DEL TEOREMA 2. Necessitarem alguns paràmetres. Definim $\ell = \sqrt{k} = \sqrt[4]{n}$, $p = \ell \log n$ i $M = (p-1)^\ell \ell!$.

Compararem un circuit monòton que resol $\text{CLICA}_{n,k}$ amb una classe especial de circuits monòtons petits que anomenem *circuits crus petits*. Un *circuit cru* és un circuit del tipus $\text{OR}_{a\in\{1,\ldots,m\}}\text{AND}_{i\neq j\in S_a}\ g_{i,j}$ on $S_1, \ldots, S_m \subseteq V$. Farem servir la notació $\text{CC}(S_1, \ldots, S_m)$ per a denotar aquest circuit cru. En particular, l'exemple de circuit que resol $\text{CLICA}_{n,k}$ del principi d'aquesta secció és un circuit cru. Direm que un circuit cru és *petit* si $m \leq M$ i a més $|S_a| \leq \ell$ per a tot $a \in \{1, \ldots, m\}$.

Transformarem qualsevol circuit monòton en un circuit cru petit inductivament (per a cada porta, des de les portes d'entrada fins a la de sortida). Demostrarem que, respecte a una classe de grafs d'entrada, a cada pas de la transformació la funció que el circuit computa només canvia una mica. D'altra banda, també demostrarem que cada circuit cru petit fa molts errors respecte a aquesta classe de grafs. Per tant, per a arribar a un circuit cru petit des d'un circuit per $\text{CLICA}_{n,k}$ s'han de fer molts passos de la construcció inductiva, la qual cosa vol dir que el circuit original té moltes portes (un nombre exponencial en n).

La classe de grafs d'entrada que estudiarem és la següent: els grafs *positius* són els grafs que consten només d'una clica de k vèrtexs i cap altra aresta —n'hi ha $\binom{n}{k}$, d'aquests. Els grafs *negatius* estan construïts a partir d'una coloració de tots els vèrtexs amb $k-1$ colors, i afegint arestes entre tots els vèrtexs de colors diferents. Aquests grafs no contenen cliques de mida k i n'hi ha $(k-1)^n$, d'aquests.

LEMA 4. *Cada circuit cru petit o bé sempre dóna «fals», o bé dóna «cert» per com a mínim la meitat dels grafs negatius.*

PROVA. Suposem que el circuit és $\text{CC}(S_1, \ldots, S_m)$. Si $m = 0$ llavors el circuit sempre dóna «fals». Si $m \geq 1$ llavors el circuit dóna «cert» per a tots els grafs negatius tals que tots els vèrtexs de S_1 tenen colors diferents. En una coloració escollida a l'atzar la probabilitat que dos vèrtexs fixats tinguin el mateix color és $\frac{1}{k-1}$. Atès que $|S_1| \leq \ell$ (segons la definició de *circuit cru petit*) la probabilitat que tots els vèrtexs de S_1 tinguin colors diferents és com a mínim

$$1 - \sum_{i\neq j\in S_1} \frac{1}{k-1} = 1 - \frac{\binom{|S_1|}{2}}{k-1} \geq 1 - \frac{(\sqrt{k})(\sqrt{k}-1)}{2(k-1)} \geq \frac{1}{2}$$

tal com calia. □

Fixem qualsevol circuit monòton C que resolgui el problema $\textsc{Clica}_{n,k}$. Aquest circuit té $\binom{n}{2}$ portes d'entrada i, diguem, T portes en total. Cada porta d'entrada $g_{i,j}$ es pot canviar pel circuit cru $\mathrm{CC}(\{i,j\})$. Cada circuit monòton és un OR o un AND de dos circuits monòtons i, per tant, només cal explicar com aproximem OR i AND de dos circuits crus petits $\mathrm{CC}(\mathcal{X})$ i $\mathrm{CC}(\mathcal{Y})$ amb un circuit cru petit, i aplicar l'aproximació recursivament.

Primer, aproximem una porta OR. El circuit cru $\mathrm{CC}(\mathcal{X} \cup \mathcal{Y})$ dóna el mateix resultat, però no és petit. Per a fer-lo petit fem servir el lema 3, que garanteix que hi ha alguna flor en el conjunt $\mathcal{X} \cup \mathcal{Y}$. L'operació *arrencar* consisteix a substituir una flor pel seu cor i repetir fins que el nombre de conjunts és com a màxim M. Així, doncs, aproximem $\mathrm{CC}(\mathcal{X} \cup \mathcal{Y})$ per $\mathrm{CC}(\text{arrencar}(\mathcal{X} \cup \mathcal{Y}))$, que és un circuit cru petit.

Segon, aproximem una porta AND. Construïm el circuit

$$\mathrm{CC}(\text{arrencar}(\{X \cup Y : X \in \mathcal{X}, Y \in \mathcal{Y}, |X \cup Y| \leq \ell\})),$$

que és un circuit cru petit.

Considerem només les classes de grafs positius i negatius, hi ha dos tipus d'error que pot introduir la construcció: falsos positius i falsos negatius. Per exemple, si els circuits $\mathrm{CC}(\mathcal{X})$ i $\mathrm{CC}(\mathcal{Y})$ donen «fals» per un graf negatiu però el circuit $\mathrm{CC}(\text{arrencar}(\mathcal{X} \cup \mathcal{Y}))$ dóna «cert», llavors la transformació de la porta OR ha introduït un fals positiu corresponent a aquest graf. El lema que hi ha a continuació, la demostració del qual ometem, fita el nombre d'errors que cada pas de l'aproximació introdueix, *i. e.*, el nombre de grafs positius per als quals canvia el resultat a negatiu i el nombre de grafs negatius per als quals canvia el resultat a positiu.

LEMA 5. *Cada pas d'aproximació introdueix com a màxim* $M^2\binom{n-\ell-1}{k-\ell-1}$ *falsos negatius i* $M^2 2^{-p}(k-1)^n$ *falsos positius.*

Només queda comparar les fites del lema 5 amb les del lema 4. Si el circuit cru petit que obtenim al final torna sempre «fals» llavors els passos d'aproximació han introduït $\binom{n}{k}$ falsos negatius i com que cada pas pot introduir-ne com a màxim $M^2\binom{n-\ell-1}{k-\ell-1}$, el nombre de passos és com a mínim

$$\frac{\binom{n}{k}}{M^2\binom{n-\ell-1}{k-\ell-1}} \geq \frac{(n/k)^\ell}{M^2} \geq n^{\sqrt[8]{n}/12}.$$

Altrament, el nombre de falsos positius és com a mínim $(k-1)^n/2$ i com que cada pas pot introduir-ne com a màxim $M^2 2^{-p}(k-1)^n$, el nombre de passos és com a mínim $2^{p-1}/M^2 \geq n^{\sqrt[8]{n}/3}/2 \geq n^{\sqrt[8]{n}/12}$. □

En el seu temps aquest teorema va generar un gran optimisme de cara a la resolució de la conjectura general. El problema semblava gairebé resolt. Una idea per a completar l'argument de Razborov i tancar el problema seria demostrar que tots els problemes monòtons a P tenen circuits monòtons de mida polinòmica, o sigui que si hi ha un circuit no monòton per a un problema monòton llavors es pot construir un circuit monòton només una mica més gran. Malauradament, aquesta conjectura va ser refutada el mateix any pel mateix Razborov fent servir la mateixa tècnica per a demostrar que el problema dels aparellaments perfectes en grafs bipartits, que és un problema monòton i pertany a la classe P, no es pot resoldre amb circuits monòtons polinòmics.

5.2 *Statu quo*

Recordeu que l'objectiu és demostrar que NP no està inclosa en P/poly, que és més fort que demostrar que NP no està inclosa en P. L'any 1983, Ajtai [2] i Furst, Saxe i Sipser [8] van fer el primer pas, demostrant que hi ha problemes a NP que no es poden resoldre amb circuits de portes AND, OR i NOT de profunditat *constant* (*i. e.*, independent de la mida de l'entrada) i mida polinòmica. Razborov [19] i Smolensky [21], pocs anys després, van aconseguir ampliar una mica aquesta classe de circuits, incloent-hi també portes MOD_p, on p és un nombre primer.[4] I què passa si fem servir portes MOD_m on m és un mòdul compost? Aquí és on el progrés s'ha quedat encallat. Sorprenentment, encara ara no podem descartar que tots els problemes de NP es puguin resoldre amb aquest tipus de circuits. És més, durant més de vint anys ningú no sabia com demostrar que hi ha problemes a NEXP (la versió indeterminista d'EXP) que no es poden resoldre amb aquests circuits. Però, finalment, Williams [23] ho va aconseguir l'any 2011.

Des del punt de vista de l'estudi de màquines, es podria pensar que aquesta situació és una mica preocupant perquè ja fa temps que sabem que $\mathrm{P} \subsetneq \mathrm{EXP}$ (fent servir la tècnica de la diagonalització). El gran avenç del resultat de Williams és que aconsegueix demostrar fites inferiors per a la mida de circuits *no uniformes*, on les tècniques dels anys vuitanta s'havien quedat estancades. En conclusió, contràriament a la intuïció que és més fàcil raonar sobre circuits que sobre màquines, els resultats de moment semblen més febles en el camp dels circuits. Tanmateix, esperonada pel resultat de Williams, que està basat en una innovadora connexió entre algoritmes exponencials no trivials i fites inferiors per a la mida de circuits, m'atreviria a especular que aquest *statu quo* potser canviarà d'aquí a pocs anys.

6 La carta de Gödel a Von Neumann

Encara que el problema «P *versus* NP» va ser plantejat formalment l'any 1971, hi ha un document de l'any 1956 que ja proposa una versió d'aquest problema.

4 Aquestes portes es poden simular amb circuits de portes AND, OR i NOT de mida polinòmica, tot i que no de profunditat constant. Per tant, encara estem parlant d'un subconjunt de P/poly.

És una carta que va escriure Gödel a von Neumann durant l'últim any de vida de Von Neumann, quan ja era a l'hospital. El text d'aquesta carta es va fer públic l'any 1988. No se sap si Von Neumann li va respondre o, ni tan sols, si va tenir l'oportunitat de considerar la pregunta de Gödel.

Aquí presentem una traducció al català de la carta, l'original de la qual està escrit en alemany. La nostra traducció parteix de la traducció en anglès publicada a [20].

Princeton, 20 de març de 1956
Benvolgut senyor Von Neumann,
Amb la més gran tristor m'he assabentat de la seva malaltia. No m'esperava gens la notícia. En Morgensten em va parlar, ja l'estiu passat, d'algun símptoma de debilitat, però en aquell moment va considerar que no seria de gran importància. He sentit dir que ha seguit un tractament radical en els últims mesos i estic content que aquest tractament hagi tingut l'èxit desitjat i que ja es trobi millor. Espero i desitjo que el seu estat millori encara més i que els nous descobriments mèdics, si és possible, el duguin a la recuperació absoluta.

Atès que, com he sentit dir, ja es troba més fort, m'agradaria permetre'm d'escriure'l sobre un problema matemàtic, del qual m'interessa molt la seva opinió: òbviament es pot construir fàcilment una màquina de Turing que, per a cada fórmula F del càlcul de predicats i cada nombre natural n, permet decidir si existeix alguna demostració de F de longitud n (longitud = nombre de símbols). Sigui $\Psi(F,n)$ el nombre de passos que requereix aquesta màquina i sigui $\phi(n) = \max_F \Psi(F,n)$. La pregunta és com creix $\phi(n)$ per a la màquina òptima. Es pot demostrar que $\phi(n) \geq k \cdot n$. Si realment existís una màquina amb $\phi(n) \sim k \cdot n$ (o fins i tot $\sim k \cdot n^2$), això tindria conseqüències de la màxima importància. Efectivament, implicaria que tot i la indecidibilitat de l'Entscheidungsproblem [problema de la decisió], *la feina intel·lectual del matemàtic en problemes sí-o-no es podria substituir completament* [nota a peu de pàgina de Gödel: *tret de la postulació d'axiomes*] *per una màquina. Al cap i a la fi, simplement s'hauria d'escollir un nombre natural n prou gran per tal que quan la màquina no produeixi cap resultat no tingui sentit pensar més en el problema. Ara a mi em sembla, tanmateix, completament dins del que és possible que $\phi(n)$ creixi tan lentament. Perquè 1) sembla que $\phi(n) \geq k \cdot n$ és l'única estimació que es pot obtenir a través d'una generalització de la demostració de la indecidibilitat de l'Entscheidungsproblem, i 2) al cap i a la fi $\phi(n) \sim k \cdot n$ (o $\sim k \cdot n^2$) només significa que el nombre de passos, comparat amb prova i error, es pot reduir des de N a $\log N$ (o $(\log N)^2$). Tanmateix, aquest tipus de reduccions dràstiques apareixen en altres problemes finits, per exemple a la computació de residus quadràtics fent servir aplicacions repetides de la llei de reciprocitat. Seria interessant saber, per exemple, la situació sobre la determinació de la primeritat d'un nombre i fins a quin punt en general el nombre de passos en problemes combinatoris es pot reduir respecte al de la cerca exhaustiva senzilla.*

No sé si ha sentit que el «problema de Post» (si existeixen graus d'irresolubilitat entre els problemes de la forma $(\exists y)\phi(x,y)$, on ϕ és recursiva) ha estat

resolt en positiu per un jove de nom Richard Friedberg. La solució és molt elegant. Malauradament, Friedberg no té intencions d'estudiar matemàtiques, sinó medicina (aparentment sota la influència del seu pare). Per cert, què en pensa dels intents de construir els fonaments de l'anàlisi sobre la teoria de tipus ramificats, que darrerament s'han posat de moda? Probablement és conscient que, en relació amb això, Paul Lorenzen ha fet progressos per a la teoria de la mesura de Lebesgue. Tanmateix, jo crec que en parts importants de l'anàlisi els mètodes de demostració impredicatius no es poden eliminar.

Estaria molt content de tenir notícies seves; i, si us plau, faci'm saber si hi ha res que pugui fer per a vostè.

Els meus més sincers desitjos, així com per a la seva dona.

Atentament,

Kurt Gödel

PS. El felicito de tot cor pel premi que el Govern americà li ha atorgat.

Gödel parla d'un problema NP-complet cèlebre. És el problema anomenat SHORT PROOF EXISTENCE, sobre l'existència de demostracions curtes. Donada una fórmula del llenguatge d'una teoria formal, un conjunt d'axiomes i un espai d'escriptura fitat, la pregunta és si existeix una demostració d'aquesta fórmula partint dels axiomes que càpiga en aquest espai. Per a aquest problema, la mida de l'entrada és l'espai donat. Així que ens interessa trobar un algoritme que tardi temps no més que polinòmic en la mida de l'espai disponible.

Gödel sembla optimista que aquest problema sigui a P i fins i tot que pugui haver-hi un algoritme quadràtic. Avui dia la majoria d'investigadors apostarien per l'altra opció, o sigui que no existeixen algoritmes polinòmics per als problemes NP-complets. Efectivament, si Gödel tingués raó, les conseqüències serien bastant més notables que en cas contrari. Una de les conseqüències, a la qual Gödel es refereix, és que podríem escriure programes per a resoldre qualsevol problema matemàtic i automatitzar tota la recerca en matemàtiques. Així que la persona que demostri que P = NP d'aquesta manera podria guanyar sis milions de dòlars perquè el seu programa podria trobar demostracions per a la resta de problemes de l'Institut Clay (assumint que tots tenen demostracions raonablement curtes).

Respecte al problema de la primeritat en particular, que és un dels exemples que posa Gödel, des de l'any 2001 sabem que sí que pertany a P [1]. Durant molts anys era un dels pocs problemes naturals que no sabíem si eren a P i que tampoc podíem demostrar que fossin NP-complets. En aquesta classe de problemes encara hi queden el problema de l'isomorfisme de grafs i el problema de la factorització de nombres enters.

Agraïments

Gràcies a Josep Pla i a Anna Puig per la seva confiança en encarregar-me la conferència a la Jornada P *versus* NP de la Facultat de Matemàtiques de la Universitat de Barcelona, que després es va convertir en aquest article. Agraeixo a Albert Atserias els seus comentaris i els seus consells sobre el contingut d'aquest article.

Referències

[1] Agrawal, M.; Kayal, N.; Saxena, N. «PRIMES is in P». *Ann. of Math.*, 160 (2) (2004), 781–793.

[2] Ajtai, M. «Σ_1^1-formulae on finite structures». *Ann. Pure Appl. Logic*, 24 (1983), 1–148.

[3] Arora, S. «Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems». *J. ACM*, 45 (5) (1998), 753–782.

[4] Björklund, A. «Determinant sums for undirected hamiltonicity». A: *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*. Washington: IEEE Computer Society, 2010, 173–182.

[5] Colbourn, C. «The complexity of completing partial latin squares». *Discrete Appl. Math.*, 8 (1984), 25–30.

[6] Cook, S. A. «The complexity of theorem-proving procedures». A: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*. Nova York: ACM, 1971, 151–158.

[7] Coppersmith, D.; Winograd, S. «Matrix multiplication via arithmetic progressions». *J. Symbolic Comput.*, 9 (3) (1990), 251–280.

[8] Furst, M.; Saxe, J.; Sipser, M. «Parity, circuits, and the polynomial time hierarchy». *Math. Systems Theory*, 17 (1984), 13–27.

[9] Garey, M. R.; Johnson, D. S. *Computers and intractability*: *A guide to the theory of NP-completeness*. Nova York: W. H. Freeman & Co., 1990.

[10] Gowers, W. T. «Razborov's method of approximations» [en línia]. Rep. tèc. 2009. `http://gowers.files.wordpress.com/2009/05/razborov2.pdf` [Consulta: 14 de maig de 2012]

[11] Held, M.; Karp, R. M. «A dynamic programming approach to sequencing problems». *J. Soc. Indust. Appl. Math.*, 10 (1) (1962), 196–210.

[12] Karp, R. M. «Reducibility among combinatorial problems». A: Miller, R. E.; Thatcher, J. W. (ed.). *Proceedings of a Symposium on the Complexity of Computer Computations*. Nova York; Londres: Plenum Press, 1972, 85–103.

[13] Kruskal, J. B. «On the shortest spanning subtree of a graph and the traveling salesman problem». *Proc. Amer. Math. Soc.*, 7 (1) (1956), 48–50.

[14] Levin, L. «Universal search problems». *Probl. Inf. Transm.*, 9 (3) (1973), 265–266.

[15] Mitchell, J. S. B. «Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems». *SIAM J. Comput.*, 28 (4) (1999), 1298–1309.

[16] Nešetřil, J.; Poljak, S. «On the complexity of the subgraph problem». *Comment. Math. Univ. Carolin.*, 26 (2) (1985), 415–419.

[17] Papadimitriou, C. *Computational complexity*. Reading, Mass.: Addison-Wesley, 1993.

[18] Razborov, A. A. «Some lower bounds for the monotone complexity of some Boolean functions». *Soviet Math. Dokl.*, 31 (1985), 354–357.

[19] Razborov, A. A. «Lower bounds on the size of bounded depth networks over a complete basis with logical addition». *Mat. Zametki*, 41 (4) (1987), 598–607. [Traduït a *Math. Notes of Academy of Sciences USSR*, 42 (4) (1987), 333–338]

[20] Sipser, M. «The history and status of the P versus NP question». A: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*. Nova York: ACM, 1992, 603–618.

[21] Smolensky, R. «Algebraic methods in the theory of lower bounds for Boolean circuit complexity». A: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*. Nova York: ACM, 1987, 77–82.

[22] Turing, A. M. «On computable numbers, with an application to the Entscheidungsproblem». *Proc. Lond. Math. Soc.*, 42, (1936), 230–265.

[23] Williams, R. «Non-uniform ACC circuit lower bounds». A: *Proceedings of 26th Annual IEEE Conference on Computational Complexity*. Washington: IEEE Computer Society, 2011, 115–125.

Departament de Matemàtica Aplicada i Anàlisi
Universitat de Barcelona
Gran Via de les Corts Catalanes, 585
08007 Barcelona
elitza.maneva@gmail.com