Material Teórica XV – Segunda parte

Temario

- Complejidad computacional y optimización

Complejidad computacional y optimización

Parte 1: Complejidad

- "Conjeturo que no hay un buen algoritmo para el problema del viajante"

  Jack Edmonds, 1967

¿Qué quiso decir con "un buen algoritmo"?

Un algoritmo es bueno si puede resolver un problema en una cantidad de tiempo que consideramos aceptable

Alan Turing inventó una máquina que llamamos máquina de Turing, que es una máquina teórica. Todo lo que puede resolverse con cualquier lenguaje lo puede resolver una máquina de Turing

Turing la definió de la siguiente manera:
"...una ilimitada capacidad de memoria obtenida en la forma de una cinta infinita marcada con cuadrados, en cada uno de los cuales podría imprimirse un símbolo. En cualquier momento hay un símbolo en la máquina; llamado el símbolo leído. La máquina puede alterar el símbolo leído y su comportamiento está en parte determinado por ese símbolo, pero los símbolos en otros lugares de la cinta no afectan el comportamiento de la máquina. Sin embargo, la cinta se puede mover hacia adelante y hacia atrás a través de la máquina, siendo esto una de las operaciones elementales de la máquina. Por lo tanto cualquier símbolo en la cinta puede tener finalmente una oportunidad."

Una máquina de Turing que es capaz de simular cualquier otra máquina de Turing es llamada una máquina universal de Turing. Las máquinas de Turing constituyen la noción informal de un método eficaz en la lógica y las matemáticas y proporcionan una definición de un algoritmo.

Algoritmos, problemas y complejidad

Algoritmo Polinomial

- Un algoritmo se dice "polinomial" si su orden de crecimiento está acotado por un polinomio en n (sin importar el grado de dicho polinomio), esto es, es del tipo $O(n^2)$ para algún número natural k

Problema polinomial

- Un problema se dice polinomial (o "fácil") si se conoce un algoritmo polinomial para determinar una solución óptima
  - Cálculo del máximo común divisor de dos números
  - Ordenación de m números
  - Inversión de una matriz de m x m
  - Cálculo de una solución de un sistema de n variables y m ecuaciones lineales

Problemas indecidibles

- No tiene algoritmos que lo resuelvan
- Ejemplo: Problema de la parada ("Halt problem")
- Dado un programa y unos datos de entrada para el mismo ¿tal programa con tales datos hará que el autómata se detenga alguna vez o no cuando se ejecute?

Una definición de Complejidad

La complejidad se define según la relación entre la cantidad de instrucciones computadas y el tamaño de la instancia

Instancia es cada una de las posibles entradas del problema

Máquina no determinística

Imaginemos una máquina con infinitos núcleos

¿Puede hacer más cosas que una PC?

Esta máquina puede resolver los mismos problemas, pero lo hace MUCHO más rápido

Es una máquina de Turing donde en cada paso se puede tomar más de una acción, estas acciones se computan en paralelo en copias de la máquina que llegó a ese estado

La máquina de Turing tiene un hilo de ejecución, la máquina no determinística tiene un árbol

¿Cuándo un algoritmo es de clase P?

- Si tiene un tiempo de ejecución polinómico
- Si existe un algoritmo de clase P que lo resuelve

Un problema de decisión está en P si una máquina de Turing determinística puede resolverlo en tiempo polinomial

- Ejemplo 1: ¿cómo determinar si un número es par o impar?

¿el último dígito es 0? SI -> ES PAR -> STOP
SINO ¿el último dígito es 2? SI -> ES PAR -> STOP
SINO ¿el último dígito es 4? SI -> ES PAR -> STOP
SINO ¿el último dígito es 6? SI -> ES PAR -> STOP
SINO ¿el último dígito es 8? SI -> ES PAR -> STOP
STOP ES IMPAR

A lo sumo 6 pasos (Independientemente del tamaño del input)

- Ejemplo 2: ¿cómo ordenar alfabéticamente un conjunto de palabras?
- El ordenamiento "burbuja", con n palabras tiene n etapas y (n – 1) comparaciones por cada etapa -> casi n2 -> tiempo de ejecución polinómico

- Ejemplo 3: ¿cómo multiplicar matrices?
- Básicamente hay dos algoritmos: el de "lápiz y papel" que para una matriz de 2x2 lleva 8 operaciones y el de Strassen que puede hacerlo en 7 segundos. A continuación vemos la comparación de los dos:

¿Cómo se le ocurrió a Strassen ese algoritmo? Era un matemático inteligente que pensó mucho. Una vez que propone el algoritmo, es fácil verificar que funciona.

Hasta ahora (por 50 años) nadie encontró una mejora sustancial. Pero frameándolo como un juego, la gente de Deepmind trabajó buscando buenos algoritmos del mismo modo que buscando buenas jugadas de ajedrez.

Podemos verlo acá:

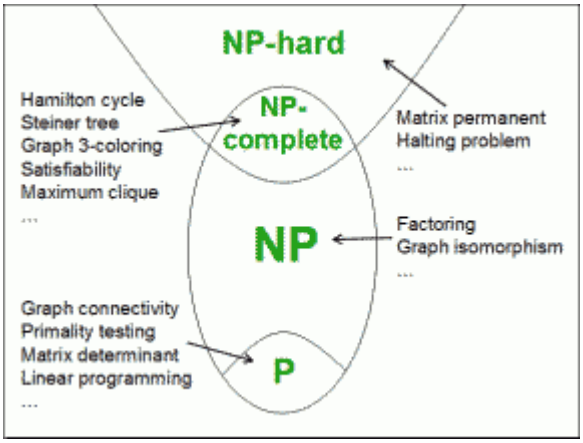
https://arstechnica.com/information-technology/2022/10/deepmind-breaks-50-year-math-record-using-ai-new-record-falls-a-week-later/

Pero sigamos con complejidad….

Cuándo un algoritmo es de Clase NP

- No determinista polinómico
- "No determinista" porque se puede resolver si uno tiene una inspiración y formula una conjetura para resolverlo
- Se puede comprobar en tiempo polinómico que la conjetura es correcta

Un problema de decisión está en NP si una máquina de Turing no-determinista puede resolverlo en tiempo polinomial



Cuándo un algoritmo es de Clase NP Completo

- Stephen Cook (1971)

THE P VERSUS NP PROBLEM

STEPHEN COOK

1. Statement of the Problem

The **P** versus **NP** problem is to determine whether every language accepted by some nondeterministic algorithm in polynomial time is also accepted by some (deterministic) algorithm in polynomial time. To define the problem precisely it is necessary to give a formal model of a computer. The standard computer model in computability theory is the Turing machine, introduced by Alan Turing in 1936 [37]. Although the model was introduced before physical computers were built, it nevertheless continues to be accepted as the proper computer model for the purpose of defining the notion of *computable function*.

- Es un problema NP específico con la propiedad de que si existe un algoritmo de clase P para resolverlo, entonces cualquier problema NP puede resolverse con un algoritmo de clase P

Algunos problemas NP-Completos:

- SAT
- Knapsack
- TSP
- Coloreo
- Máximo clique
- Sudoku
- Tetris
- Muchos otros…

Lista de 21 problemas NP-completos de Karp

- SAT (Problema de satisfacibilidad booleana, para fórmulas en forma normal conjuntiva)
  - 0-1 INTEGER PROGRAMMING (Problema de programación lineal entera)
  - CLIQUE (Problema del clique, o Problema del conjunto independiente)
    - SET PACKING (Problema del empaquetamiento de conjuntos)
    - VERTEX COVER (Problema de cobertura de vértices)
      - SET COVERING (Problema de cobertura de conjuntos)
      - FEEDBACK NODE SET (Conjunto de vértices que hay que sacar para que el grafo quede sin ciclos)
      - FEEDBACK ARC SET (Conjunto de arcos que hay que sacar para que el grafo quede DAG)
      - DIRECTED HAMILTONIAN CIRCUIT (Problema del circuito hamiltoniano dirigido)
        - UNDIRECTED HAMILTONIAN CIRCUIT (Problema del circuito hamiltoniano no dirigido)
  - 3-SAT (Problema de satisfacibilidad booleana de 3 variables por cláusula)
    - CHROMATIC NUMBER (Problema de coloreo de grafos)
      - CLIQUE COVER (Problema de la cobertura de cliques)
      - EXACT COVER (Problema de la cobertura exacta)
        - HITTING SET (formulación equivalente a Set Cover)
        - STEINER TREE (árbol generador mínimo con agregado de vértices en el grafo)
        - 3-DIMENSIONAL MATCHING (Problema del matching tridimensional)
        - KNAPSACK (Problema de la mochila)
          - JOB SEQUENCING (Problema de las secuencias de trabajo)
          - PARTITION (Problema de la partición)
            - MAX-CUT (Problema del corte máximo)

Máquina de Turing no determinística

En cada paso se puede tomar más de una acción

Estas acciones se computan en paralelo en copias de la máquina que llegó a ese estado

Reducciones

- ¿Si puedo contestar el problema de decisión del TSP puedo contestar el del camino Hamiltoniano?

El problema del camino Hamiltoniano es, simplemente, responder si existe un camino que pase por cada vértice una única vez.

HamiltonianCycle(G=(V,E))

Construct a complete weighted graph G'=(V',E') where V'=V.

n = |V|

for i = 1 to n do

for j = 1 to n do

if then w(i,j) = 1 else w(i,j) = 2

Return the answer to Traveling-Salesman(G',n).

Reducción en tiempo polinomial

Si un problema A puede "convertirse" en tiempo polinomial en un problema B decimos que B "no es más fácil" que A, esto es lógico ya que si podemos resolver B de forma eficiente lo mismo sucede con A

SAT: Primer problema NP-Completo

$$(X1 \text{ or } X2 \text{ or } \overline{X3})$$
$$(\overline{X1} \text{ or } \overline{X2} \text{ or } X3)$$
$$(\overline{X1} \text{ or } \overline{X2} \text{ or } \overline{X3})$$
$$(\overline{X1} \text{ or } X2 \text{ or } X3)$$

De ahora en adelante cualquier reducción de SAT será un problema NP-Difícil

Nota: NP-Difícil + NP = NP-Completo

Parte 2: Optimización

¿qué complejidad tiene el método simplex?

En la práctica es eficiente, pero en el peor caso la complejidad es exponencial. Esto es porque en el caso promedio se puede resolver con complejidad polinomial.

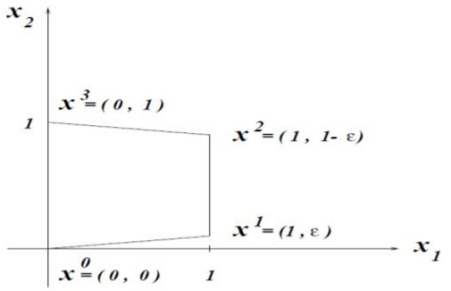
(se puede ver Klee, Victor; Minty, George J. (1972). "How good is the simplex algorithm?". In Shisha, Oved. Inequalities III (Proceedings of the Third Symposium on Inequalities held at the University of California, Los Angeles, Calif., September 1–9, 1969, dedicated to the memory of Theodore S. Motzkin). New York-London: Academic Press. pp. 159–175.)

Worst case performance of the simplex method

Klee-Minty Example:

- Victor Klee, George J. Minty, "How good is the simplex algorithm?" in (O. Shisha edited) Inequalities, Vol. III (1972), pp. 159-175.

$$
\begin{aligned}
(2\ \text{dim})\quad \min\ & -x_2 \\
\text{s.t.}\ & x_1 \geq 0 \\
& x_1 \leq 1 \\
& x_2 \geq \epsilon x_1 \qquad \left(0 < \epsilon < \frac{1}{2}\right) \\
& x_2 \leq 1 - \epsilon x_1 \\
& x_1, x_2 \geq 0
\end{aligned}
$$

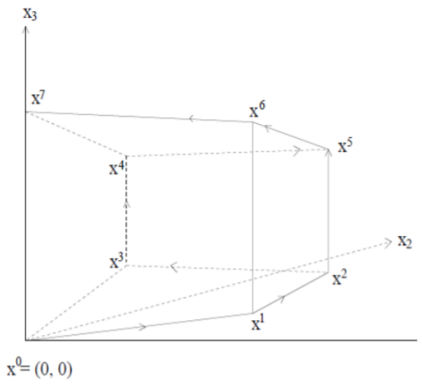


$\mathbf{x}^0 \rightarrow \mathbf{x}^1 \rightarrow \mathbf{x}^1 \rightarrow \mathbf{x}^3$ (optimal)

$2^2 - 1 = 3$ iterations

Klee-Minty Example

$$
\begin{aligned}
(3\ \text{dim})\quad \min\ & -x_3 \\
\text{s.t.}\ & x_1 \geq 0 \\
& x_1 \leq 1 \\
& x_2 \geq \epsilon x_1 \\
& x_2 \leq 1 - \epsilon x_1 \\
& x_3 \geq \epsilon x_2 \\
& x_3 \leq 1 - \epsilon x_2 \\
& x_1, x_2, x_3 \geq 0
\end{aligned}
$$



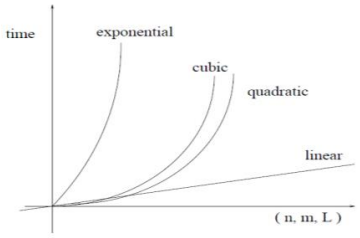
$2^3 - 1 = 7$ iterations

Klee-Minty Example

$$
\begin{aligned}
(d \text{ dim}) \quad \min \quad & -x_d \\
\text{s. t.} \quad & x_1 \geq 0 \\
& x_1 \leq 1 \\
& x_2 \geq \epsilon x_1 \\
& x_2 \leq 1 - \epsilon x_1 \\
& \vdots \\
& x_d \geq \epsilon x_{d-1} \\
& x_d \leq 1 - \epsilon x_{d-1} \\
& x_i \geq 0
\end{aligned}
$$

$2^d - 1$ iterations

Hence, in theory, the simplex method is not a polynomial-time algorithm. It is an *exponential time* algorithm!



Método del elipsoide

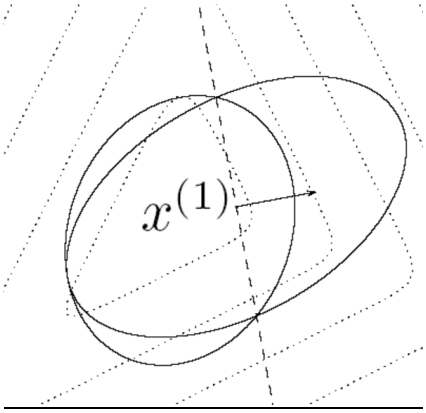
En 1979 Leonid Khachiyan desarrolla el método del elipsoide con complejidad polinomial para resolver problemas de programación lineal

- Para resolver
  max cX st. Ax ≤ b, x ≥ 0

  podemos encontrar una solución de
  Ax ≤ b , x ≥ 0, yA ≥ c , y ≥ 0, cx ≥ yb

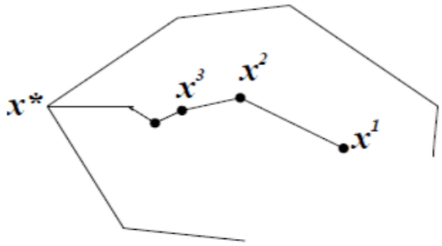
- Resolver un sistema de desigualdades no es más difícil, desde el punto de vista teórico, que sólo reconocer si un sistema tiene soluciones

- El método del elipsoide da una forma teóricamente satisfactoria de reconocer si un sistema de desigualdades tiene solución
  Este algoritmo no pudo ser aplicado en la práctica pero tiene un gran valor teórico ya que demuestra que la programación lineal está en P

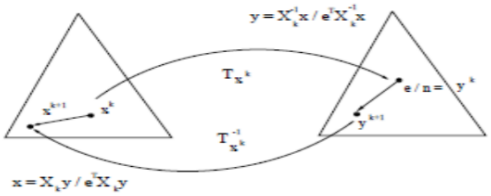


Algoritmo de Karmarkar

Inspirado en el método del elipsoide, Narendra Karmarkar desarrolla en 1984 un algoritmo de punto interior con complejidad polinomial que tiene utilidad práctica



Karmarkar's algorithm



$$(P)\quad \begin{array}{ll} \min & \mathbf{c}^T\mathbf{x} \\ \text{s. t.} & \mathbf{A}\mathbf{x} = 0 \\ & e^T\mathbf{x} = 1 \\ & \mathbf{x} \geq 0 \end{array}$$

$$\begin{array}{ll} \min & \frac{\mathbf{c}^T\mathbf{X}_k\mathbf{y}}{e^T\mathbf{X}_k\mathbf{y}} \\ \text{s. t.} & \mathbf{A}\mathbf{X}_k\mathbf{y} = 0 \\ & e^T\mathbf{y} = 1 \\ & \mathbf{y} \geq 0 \end{array}$$

$$(P')\quad \begin{array}{ll} \min & (\mathbf{c}^T\mathbf{X}_k)\mathbf{y} \\ \text{s. t.} & \left[\begin{smallmatrix}\mathbf{A}\mathbf{X}_k \\ e^T\end{smallmatrix}\right]\mathbf{y} = \left[\begin{smallmatrix}0 \\ 1\end{smallmatrix}\right] \\ & \mathbf{y} \geq 0 \end{array}$$

$$\begin{array}{ll} \mathbf{c}^k & = \mathbf{X}_k\mathbf{c} \\ \mathbf{B}_k & = \left[\begin{smallmatrix}\mathbf{A}\mathbf{X}_k \\ e^T\end{smallmatrix}\right] \end{array}$$

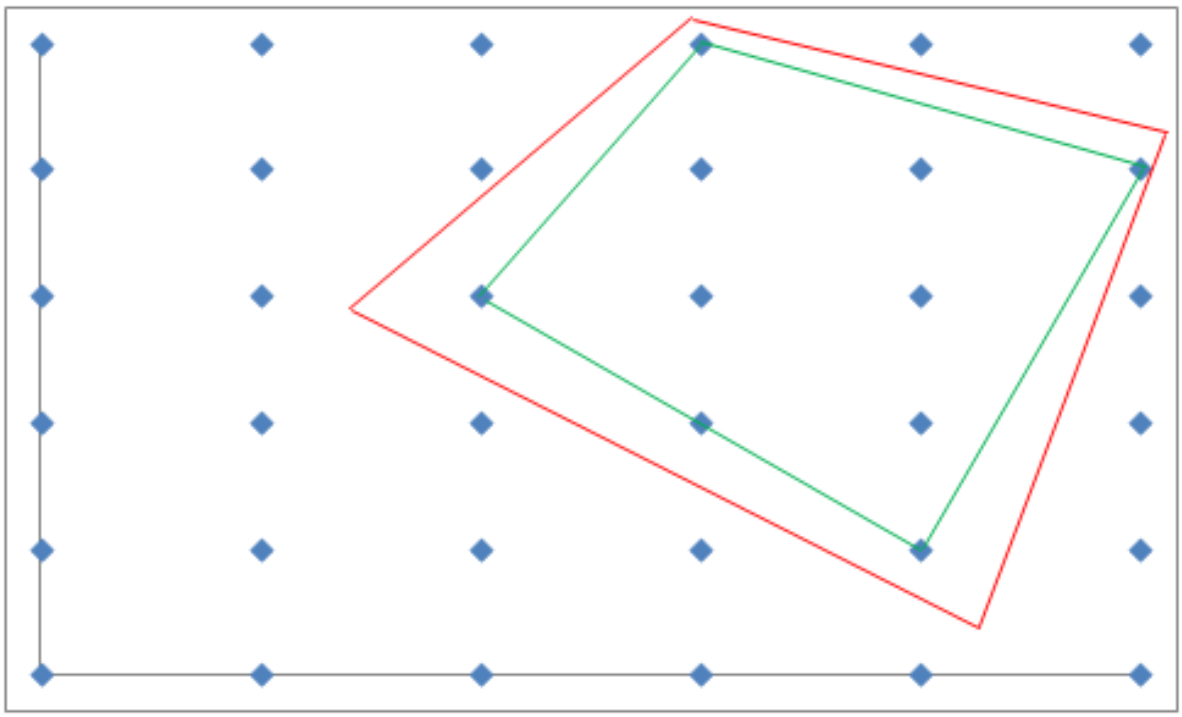
Simplex nos motiva a repensar las métricas de complejidad

Tenemos un algoritmo de complejidad exponencial que en la práctica es similar a los algoritmos polinómicos conocidos

Parte 3: Optimización Entera

¿Es difícil resolver un problema de PLE?

Cápsula convexa



Ralph Gomory, en 1950 desarrolla un procedimiento que agrega restricciones válidas a un recinto quitando soluciones fraccionarias

Este procedimiento converge a la cápsula convexa en tiempo finito pero se lo considera ineficiente

En 1990 se lo combina con Branch & Cut con muy buenos resultados, hoy en día se lo usa de alguna forma en todo software comercial

¿Es difícil hallar la cápsula convexa?

En muchos casos tiene una cantidad exponencial de restricciones que la define y hallar cada una de estas es difícil (NP-Difícil)