Algoritmos de Aproximación

Tópicos Avanzados en Teoría de la Computación

Ignacio Morales

Departamento de Ciencias de la Computación
Pontificia Universidad Católica de Chile

24 de Junio de 2016

Motivación

Principal Motivación:

Encontrar una clase de complejidad que tenga un conjunto de problemas completos con la siguiente propiedad:

Si ocurre que "_____" entonces $P = NP$

donde "_____" es algo relacionado con algoritmos de aproximación.

Comentarios:

- Aunque suene "sencillo", NO lo es.
- De pasada, Papadimitriou aprovecha de juntar 2 grandes áreas.

Generalización de problemas

Definición (Function Problem)

Problemas que requieren una respuesta elaborada (más que "si" o "no")

Observaciones:

- Es fácil ver que caen dentro los problemas de computación, optimización e incluso los de decisión si es que se toman como problemas con output $\{0,1\}$.

Generalización de problemas

Definición (Function Reduction)

Un problema de función A reduce a otro B si existen funciones de string R y S, ambas computables en espacio logarítmico. Además, para todo string x, z se sigue que:

- Si x es una instancia de A, entonces $R(x)$ es instancia de B.
- Si z es un output correcto de $B(R(x))$ entonces $S(z)$ es un output correcto de $A(x)$.

Formalización de Optimización

Definición (Optimization Problem)

Problemas tales que para cada instancia x, se tiene un conjunto de **soluciones factibles** $F(X)$, y por cada solución $s \in F(x)$ tenemos un costo entero positivo $c(s)$.
El costo óptimo se define como:

$$\text{OPT}(x) = \min_{s \in F(x)} c(s) \quad \text{, (o maximizar)}$$

Algoritmo

Definición (Approximation algorithm)

Sea M un algoritmo tal que para una instancia x entrega una solución factible $M(x) \in F(x)$.
Decimos que M es una ε-aproximado, donde $\varepsilon > 0$, si para todo x tenemos que:

$$\frac{|c(M(x)) - \mathrm{OPT}(x)|}{\max\{\mathrm{OPT}(x), c(M(x))\}} \leq \varepsilon$$

Intuición: Un algoritmo es ε-aproximado si el "error relativo" esta acotado por ε para toda las soluciones de $F(x)$.

Algoritmo

Observemos que por la definición, se arrastran las siguientes consecuencias:

- La constante ε siempre esta entre 0 y 1.
- ¿Que ocurre con maximizar y minimizar?

Algoritmo

Observemos que por la definición, se arrastran las siguientes consecuencias:

- La constante ε siempre esta entre 0 y 1.
- ¿Que ocurre con maximizar y minimizar?
  - Maximizar: el algoritmo entrega soluciones que no son más chicas que $1 - \varepsilon$ veces el óptimo.
  - Minimizar: el algoritmo entrega soluciones que no a lo más $\frac{1}{1-\varepsilon}$ veces el óptimo.

Umbral

Definición (Approximation Threshold)

El umbral de aproximación de un problema de optimización A se define como la cota inferior mas grande para todos los $\varepsilon > 0$ tales que exista un algoritmo ε-aproximado que corra en tiempo polinomial para A.

Umbral

Definición (Approximation Threshold)

El umbral de aproximación de un problema de optimización A se define como la cota inferior mas grande para todos los $\varepsilon > 0$ tales que exista un algoritmo ε-aproximado que corra en tiempo polinomial para A.

Intuición: el umbral es como el ínfimo del conjunto de ε's que tienen algorimos aproximados.

Preguntas

¿Qué pasa si el umbral es 1?
¿Qué pasa si el umbral esta en (0,1)?
¿Y si es 0?

PTAS

¿Qué pasa cuando me acerco a 0?

PTAS

¿Qué pasa cuando me acerco a 0?

Definición (Polynomial-time approximation scheme)

Un esquema de algoritmo de aproximación en tiempo polinomial (PTAS) es un esquema que para todo $\varepsilon > 0$ entrega un algoritmo polinomial ε-aproximado.

Noción de Reducción

¿Qué propiedades queremos?

Noción de Reducción

¿Qué propiedades queremos?

Dado un ε-algoritmo polinomial $\rightarrow$ algún ε'-algoritmo polinomial

Noción de Reducción

¿Qué propiedades queremos?

Dado un ε-algoritmo polinomial $\rightarrow$ algún ε'-algoritmo polinomial

Al menos sabemos que un problema de optimización es un problema de función.

TENEMOS POR DONDE PARTIR!!!

Recordemos

Definición (Function Reduction)

Un problema de función A reduce a otro B si existen funciones de string R y S, ambas computables en espacio logarítmico. Además, para todo string x, z se sigue que:

- Si x es una instancia de A, entonces $R(x)$ es instancia de B.
- Si z es un output correcto de $B(R(x))$ entonces $S(z)$ es un output correcto de $A(x)$.

Noción de Reducción

$\mathcal{L}$-Reduction

Supongamos que A y B son problemas de optimización. Una $\mathcal{L}$-reducciónde A a B es un par de funciones R y S, calculables en espacio logarítmico y tiempo polinomial respectivamente, y que cumplen:

- Si x es una instancia de A con costo óptimo $\mathrm{OPT}_A(x)$, entonces $R(X)$ es una instancia de B con un óptimo que satisface

$$\mathrm{OPT}_B(R(x)) \leq \alpha \mathrm{OPT}_A(x), \tag{1}$$

  donde α es una constante positiva.

- Si s es una solución factible de $B(R(x))$, entonces $S(s)$ es una solución factible de $A(x)$ tal que

$$|\mathrm{OPT}_A(x) - c(S(s))| \leq \beta |\mathrm{OPT}_B(R(x)) - c(s)|, \tag{2}$$

  donde β es una constante positiva.

Propiedades de $\mathcal{L}$-reducción

Proposición

Si (R, S) es una $\mathcal{L}$-reducciónde A a B y (R', S') es una $\mathcal{L}$-reducciónde B a C, entonces $(R \circ R', S' \circ S)$ es una $\mathcal{L}$-reducciónde A a C

Proposición

Si existe una $\mathcal{L}$-reducción(R, S) de A a B con constantes α y β, y existe un algoritmo polinomial ε-aproximado para B, enntonces existe un $\frac{\alpha\beta\varepsilon}{1-\varepsilon}$-aproximado para A.

Observación: Por la segunda proposición, se puede deducir que si existe un $\mathcal{L}$-reducciónde A a B, se tiene que:

- Si existe un PTAS para B, entonces existe uno para A.
- Si no existe un PTAS para A, entonces no existe uno para B.

Clases de Complejidad

Teorema de Fagin

Todas las propiedades de teoría de grafos en NP pueden ser expresadas en lógica de segundo orden existencial ($\exists$LSO).

Esto da la PRIMERA noción de NP que no necesita un modelo de máquina por detrás.

Clases de Complejidad

Teorema de Fagin

Todas las propiedades de teoría de grafos en NP pueden ser expresadas en lógica de segundo orden existencial ($\exists$LSO).

Esto da la PRIMERA noción de NP que no necesita un modelo de máquina por detrás.

Ejemplos

- CNFSAT: $\exists T\ \forall x\ \ [C(x) \rightarrow \exists y(P(x,y) \wedge T(y)) \vee (N(x,y) \wedge \neg T(y))]$
- 3SAT: $\exists T\ \forall x\ [C(x) \rightarrow$
$\forall w\ \forall\ y\ \forall z\ \ (P_1(x,w) \wedge P_2(x,y) \wedge P_3(x,z) \rightarrow T(w)\ \ \vee T(y) \vee T(z)) \vee$
$\cdots \vee (N_1(x,w) \wedge N_2(x,y) \wedge N_3(x,z) \rightarrow \neg T(w) \vee \neg T(y) \vee \neg T(z))]$

Ejemplo: cómo codificar una fórmula lógica

Sea φ una fórmula en CNF sobre las variables proposicionales $var = \{x_1, \ldots, x_n\}$, con $m \in \mathbb{N}^+$ cláusulas.
Sea $\mathcal{V} = \{C(\cdot), P(\cdot,\cdot), N(\cdot,\cdot)\}$ un vocabulario con una relación unaria y dos binarias. Definimos la $\mathcal{V}$-estructura $\mathfrak{A}_\varphi$ como:

$$\mathfrak{A}_\varphi = \left\langle \{x_1, \ldots, x_n\} \cup \{c_1, \ldots, c_m\}, C^{\mathfrak{A}_\varphi}, P^{\mathfrak{A}_\varphi}, N^{\mathfrak{A}_\varphi} \right\rangle,$$

donde

- $C^{\mathfrak{A}_\varphi} = \{c_1, \ldots, c_m\}$ es el conjunto de cláusulas,
- $P^{\mathfrak{A}_\varphi} = \{(x, c) : c \in C \text{ y } x \in var \text{ está positiva en } c\}$ y
- $N^{\mathfrak{A}_\varphi} = \{(x, c) : c \in C \text{ y } x \in var \text{ está negativa en } c\}$.

la cual representa de forma inequívoca a φ.

Destaquemos:

- Un problema en NP (de grafo) puede ser expresado como una formula existencial de la forma

$$\exists S \ \forall x \exists y \phi(x, y, S).$$

- Un problema en NP-estricto (SNP) es tal que puede ser expresado de la forma:

$$\exists S \ \forall x \phi(x, y, S).$$

donde ϕ es una fórmula libre de cuantificadores en lógica de primer orden.

Clases de Complejidad

Ahora, si tomamos un problema en NP, podemos expresar un problema de maximización asociado como:

$$\max_{S} |\{x : \exists y \phi(x, y, S)\}|.$$

y podemos definir la clase MAXNP como todos los problemas de maximización asociados a un problema en NP.

Pregunta

¿Se puede hacer lo mismo para SNP?

Clases de Complejidad

Ahora, si tomamos un problema en NP, podemos expresar un problema de maximización asociado como:

$$\max_S |\{x : \exists y \phi(x, y, S)\}|.$$

y podemos definir la clase MAXNP como todos los problemas de maximización asociados a un problema en NP.

Pregunta

¿Se puede hacer lo mismo para SNP?

Existe un pequeño detalle, NP es cerrado bajo reducciones (normales), pero SNP no.

Clases de Complejidad

Al tomar un problema Π en SNP (es de decisión), podemos generar su problema de maximización relacionado al igual que antes,

$$\max_{S} |\{x : \ \phi(x, S)\}|.$$

Pero ahora, definimos la clase MAXSNP_0 como todos los problemas de maximización relacionados con problemas en SNP.

Clases de Complejidad

Al tomar un problema Π en SNP (es de decisión), podemos generar su problema de maximización relacionado al igual que antes,

$$\max_{S} |\{x : \ \phi(x, S)\}|.$$

Pero ahora, definimos la clase MAXSNP_0 como todos los problemas de maximización relacionados con problemas en SNP.

Y ya tenemos todos los ingredientes!!!

Clase MAXSNP

Definición (MAXSNP)

Definimos la clase MAXSNP como:

$$\text{MAXSNP} = \{\Pi : \Pi \text{ es un problema de optimización y } \exists \Pi' \in \text{MAXSNP}_0 \text{ con una } \mathcal{L}\text{-reducción de } \Pi \text{ a } \Pi'\}$$

Clase MAXSNP

Definición (MAXSNP)

Definimos la clase MAXSNP como:

$$\text{MAXSNP} = \{\Pi : \Pi \text{ es un problema de optimización y } \exists \Pi' \in \text{MAXSNP}_0 \text{ con una } \mathcal{L}\text{-reducción de } \Pi \text{ a } \Pi'\}$$

Preguntas

¿Es cerrado bajo $\mathcal{L}$-reducción?
¿Tiene problemas completos?

MAXSNP-Completitud

Asumiendo que existe un conjunto de problemas NP completos, ¿que podemos deducir?

MAXSNP-Completitud

Asumiendo que existe un conjunto de problemas NP completos, ¿que podemos deducir?

Proposición

Si un problema completo en MAXSNP tiene un PTAS, entonces todos los problemas en MAXSNP tienes un PTAS.

MAXSNP-Completitud

Asumiendo que existe un conjunto de problemas NP completos, ¿que podemos deducir?

Proposición

Si un problema completo en MAXSNP tiene un PTAS, entonces todos los problemas en MAXSNP tienes un PTAS.

La pregunta natural es:
¿Existen problemas MAXSNP completos bajo $\mathcal{L}$-reducción?

Problemas MAXSNP completos

Algunos problemas conocidos que son completos son:

- MAX3SAT
- 4-DEGREE INDEPENDET SET (MAX)
- 4-DEGREE NODE COVER (MIN)
- MAXCUT

Problemas MAXSNP completos

Algunos problemas conocidos que son completos son:

- MAX3SAT
- 4-DEGREE INDEPENDET SET (MAX)
- 4-DEGREE NODE COVER (MIN)
- MAXCUT

Aun queda la pregunta:

¿Estos problemas admiten un PTAS?

¿Inaproximabilidad?

Definición: ((log n, 1)-Restricted Verifier)

Un verificador $(\log n, 1)$-restringido V es una maquina de Turing (probabilística) que corre en tiempo polinomial y que tiene acceso a una entrada x, una cadena r compuesta de $O(\log |x|)$ *bits random* y una prueba Π.
Dependiendo de la entrada x, la cadena *random* r y la prueba Π, el verificador V aceptara o rechazará la entrada x.
Aunque el verificador tiene acceso completo a la entrada x, el acceso a la prueba Π es muy limitado, V solamente examina un numero constante de *bits*.

¿Inaproximabilidad?

Teorema PCP

Un lenguaje L está en NP si y solo si L es decidible por un verificador $(\log n, 1)$-restringido.
Siendo decidible según:

- Si $x \in$ L, entonces existe Π tal que $Pr[V^{\Pi}(x) = 1] = 1$.
- Si $x \notin$ L, entonces para todo Π se tiene que $Pr[V^{\Pi}(x) = 1] \leq \frac{1}{2}$.

¿Inaproximabilidad?

Teorema

Si existe un PTAS para MAX3SAT, entonces P=NP

¿Inaproximabilidad?

Teorema

Si existe un PTAS para MAX3SAT, entonces P=NP

Esquema General:

- Tomar un problema en NP
- Usar el teorema de PCP para tener el verificador restringido
- Generar una formula ϕ que sea satistacible solo si la maquina acepta la entrada
- Pasar la formula a 3-CNF y luego usar el PTAS
- Concluir en base a imposibilidad en la parte donde rechaza, de manera que aunque haya probabilidad de error, igual concluyo que es rechazada.

¿Inaproximabilidad?

Idea de la demostración:

Sea L en NP y por lo tanto cumple el Teorema PCP

- Queremos construir una expresión booleana que exprese cuando la computación de V termine en "si".
- Supongamos que durante el computo de V se usan $|r| = c \log n$ bits *random*, y que se accede a solo d accesos de Π, digamos $\Pi_1^r, ..., \Pi_d^r$.
- Notar que todo esta determinado en la computación, excepto los Π's y r.
- Podemos escribir el output de V como una expresión booleana que use los $\Pi_1^r, ..., \Pi_d^r$ como variables.

¿Inaproximabilidad?

Continuación:

- Luego, podemos pasar esa expresión a 3-CNF con una cantidad $K(d)$ contante de clausulas, denotada por ϕ_r
- Repitiendo esto para las $2^{c \log n} = n^c$ secuencias de variables random para V, terminamos con $K(d)n^c$ clausulas.

Sea $\phi = \bigwedge_r \phi_r$ Ahora tenemos que:

- Si $x \in L$ entonces existe la asignación que hace verdadero a los ϕ_r para cualquier r, ya que la probabilidad de éxito es 1. Por lo tanto ϕ es verdadero.
- Si $x \notin L$ entonces la probabilidad de error es menor a $1/2$, lo que se traduce en que toda asignación debe generar al menos una clausula negativa en al menos la mitad de los ϕ's. Esas son cn y eso deja una fracción de al menos $\frac{1}{2K}$ clausulas negativas.

¿Inaproximabilidad?

Continuación: Ahora utilizamos que existe un PTAS en 3SAT, entonces usamos $\varepsilon = \frac{1}{4K}$.

$$\frac{|c(M(x)) - \mathrm{OPT}(x)|}{\max\{\mathrm{OPT}(x), c(M(x))\}} \leq \varepsilon$$

$$\frac{\mathrm{OPT}(x) - c(M(x))}{\mathrm{OPT}(x)} \leq \frac{1}{4K}$$

$$\mathrm{OPT}(x)\left(1 - \frac{1}{4K}\right) \leq c(M(x))$$

-Si el algoritmo entrega una asignación que satisface una fracción mayor a $1 - \frac{1}{2K}$ de las clausulas, entonces sabemos que $x \in L$.
-Si entrega una que es menor, tenemos que el óptimo ya no puede ser todas las clausulas, por lo que $x \notin L$