Análisis de Algoritmos
Unidad 5: Clasificacion de Problemas

Ing. Matias Valdenegro T.

Universidad Tecnológica Metropolitana

31 de enero de 2012

Preliminares

Teoria de la Complejidad Computacional

Algoritmos Aproximados

Preliminares

Teoria de la Complejidad Computacional

Algoritmos Aproximados

Introduccion

Tiempo Polinomial

Un algoritmo se denomina de tiempo polinomial si:

$$W(n) \in O(n^k), \qquad k > 0$$

Esto significa que si la complejidad del peor caso del algoritmo esta en $O(n^k)$, con un k fijo y positivo, el algoritmo se denominara de tiempo polinomial.

Todos los algoritmos que hemos visto en el curso son de tiempo polinomial.

Ejemplo

- Ordenamiento: $O(n \log n) \in O(n^2)$.
- Busqueda Lineal: $O(n) \in O(n)$.
- Busqueda Binaria: $O(\log n) \in O(n)$.
- Etc.

Algoritmos Deterministicos

Informalmente, es un algoritmo que se comporta de manera predecible. Matematicamente, esto significa que es una funcion, la cual tiene un unico valor de salida para cada valor de entrada.

Formalmente

Un algoritmo deterministico es una maquina de estados. La propiedad fundamental es que el estado actual determina cual sera el proximo estado, y por ende el algoritmo siempre ejecuta los mismos pasos dado el mismo valor de la entrada.

Algoritmos No-Deterministicos

Un algoritmo no-deterministico es un algoritmo que se comporta de manera impredecible, donde el estado actual puede dar paso a multiples posibles estados siguientes. Esto se logra a traves de:

- Automatas Finitos No Deterministicos.
- Aleatoriedad.
- Influencias externas (Entrada del usuario, datos almacenados en memoria externa, etc).
- El cerebro humano.

Problemas

En esta unidad se analizan problemas que no hemos visto hasta ahora, pero son problemas de interes general:

- Problemas de Optimizacion.
- Problemas de Optimizacion Combinatorial.
- Planificacion.

Satisfactibilidad Booleana (SAT)

Este problema consiste en determinar los valores de las variables en una formula booleana (con operaciones AND, OR, NOT, etc) de tal forma que la formula evalue al valor verdadero (true), o indicar que ningun valor de las variables hace cumplir dicha condicion. Por ejemplo:

$$E = (x_1 \vee \bar{x_2} \vee \bar{x_3}) \wedge (x_1 \vee x_2 \vee x_4)$$

Es posible encontrar valores para x_1, x_2, x_3 y x_4 tal que E evalue a verdadero?

Satisfactibilidad Booleana (SAT)

Este problema consiste en determinar los valores de las variables en una formula booleana (con operaciones AND, OR, NOT, etc) de tal forma que la formula evalue al valor verdadero (true), o indicar que ningun valor de las variables hace cumplir dicha condicion. Por ejemplo:

$$E = (x_1 \vee \bar{x_2} \vee \bar{x_3}) \wedge (x_1 \vee x_2 \vee x_4)$$

Es posible encontrar valores para x_1, x_2, x_3 y x_4 tal que E evalue a verdadero?
Si. Por ejemplo, $x_1 = true$, o
$x_1 = true, x_2 = true, x_3 = true, x_4 = true$

Caminos de Euler y Ciclos Hamiltonianos

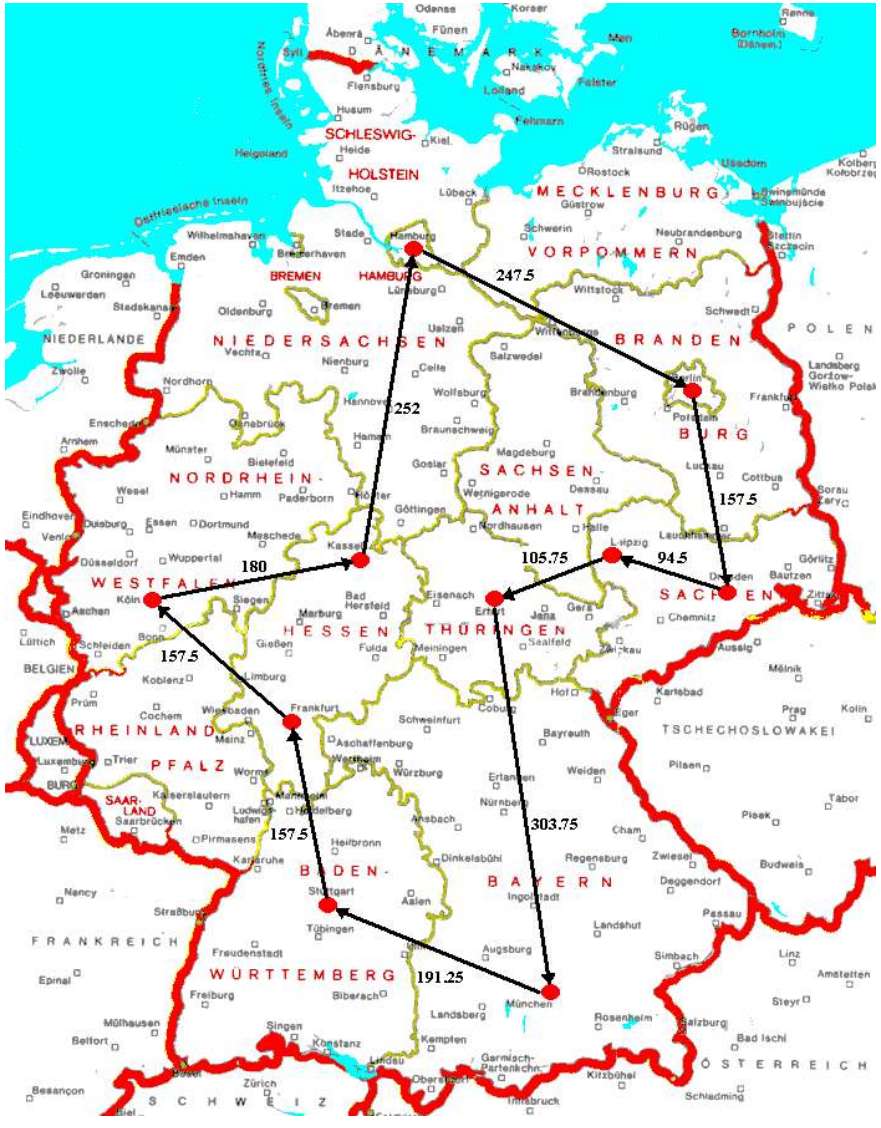
Recordando:

- Camino de Euler: Recorrido de las aristas de un grafo donde cada arista se visita solo una vez.
- Ciclo Hamiltoniano: Recorrido de los vertices de un grafo donde cada vertice se visita solo una vez.

Problema del Vendedor Viajero

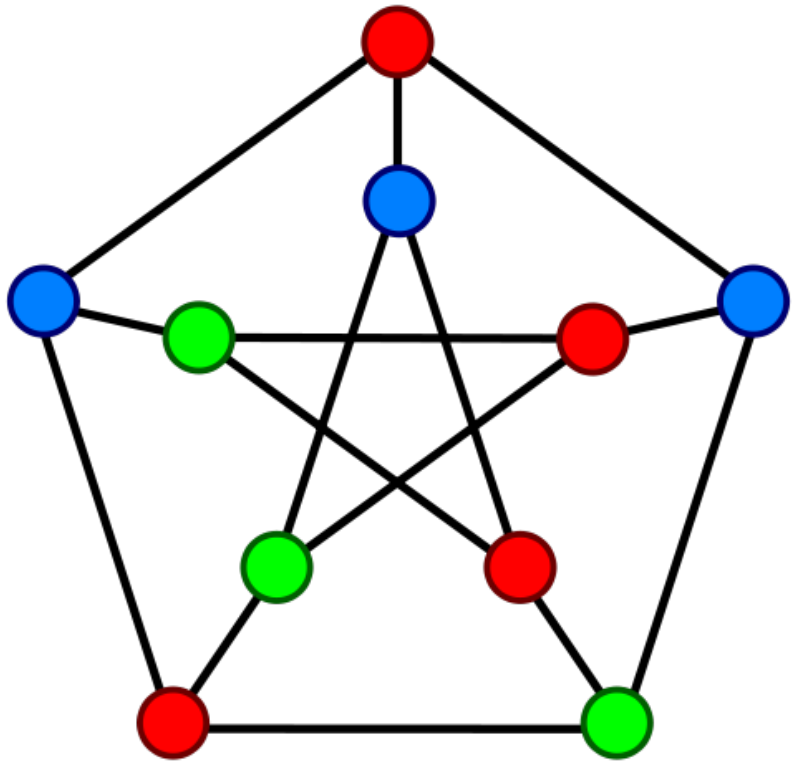
El PVV (Traveling Salesman Problem) es un caso del problema del Ciclo Hamiltoniano, cuando se aplica en un grafo ponderado. El problema consiste en encontrar un ciclo hamiltoniano de costo total minimo.

$$Costo = \text{mín} \sum_{i \in C} V_i$$


247.5
252
157.5
105.75
94.5
180
157.5
157.5
303.75
191.25
SCHLESWIG-
HOLSTEIN
MECKLENBURG
VORPOMMERN
BREMEN
HAMBURG
NIEDERSACHSEN
BRANDEN
BURG
NORDRHEIN-
SACHSEN
ANHALT
WESTFALEN
SACHSEN
HESSEN
THÜRINGEN
RHEINLAND
PFALZ
SAAR-
LAND
BADEN-
WÜRTTEMBERG
BAYERN
NIEDERLANDE
BELGIEN
LUXEMB.
FRANKREICH
SCHWEIZ
ÖSTERREICH
POLEN
TSCHECHOSLOWAKEI
DÄNEMARK

Coloreado de Grafos

El coloreado de grafos es una asignacion de colores a cada vertice de un grafo, de tal manera que dos vertices adyacentes no compartan el mismo color.



Coloreado de Grafos

De este problema se desprenden varios sub-problemas:

- Cuantos colores se requieren como minimo para colorear un grafo?
- Es posible colorear un grafo con a lo mas K colores?
- Dado un numero de colores K, cuantos coloreados diferentes existen para un grafo?

Planificacion de Tareas

Planificar tareas es un problema generico que se plantea de la siguiente forma:

Definicion

Dado un conjunto de tareas T_i, una duracion de cada tarea D_i, y un conjunto de restricciones en la duracion o relacion de dependencia entre las tareas. Es posible asignar las tareas de tal forma:

- Minimizar la duracion total.
- Cumplir cierta meta (deadline) tal que la duracion total sea $\leq D$.

Planificacion de Tareas

Posee varios sub-problemas, entre ellos:

Problema de la Mochila

Supongamos que tenemos una mochila que puede cargar a lo mas C unidades de peso, y objetos de tamaño s_i, cada objeto tiene asociada una utilidad p_i. El problema es maximizar la utilidad total de cualquier subconjunto de objetos que tenga peso total $\leq C$.

Empacado

Supongamos que tenemos un numero ilimitado de cajones de capacidad unitaria, y objetos de tamaño s_i, donde los $s_i \in [0, 1]$. Se debe encontrar un empacado de los objetos que minimice el numero de cajones usados.

Programacion Lineal Entera/Booleana

Es el problema clasico de programacion lineal, pero con la restriccion de que las variables x_i pueden tomar solo valores enteros o booleanos, incluso con combinaciones de ellos (real/entero, real/booleano, etc):

Max/min: $a_i x_1 + a_2 x_2 + \cdots + a_n x_n$
Sujeto a:

$$
\begin{gathered}
b_1 1x1 + b12x_2 + \cdots + b1nx_n \leq c_1 \\
b_2 1x1 + b22x_2 + \cdots + b2nx_n \leq c_2 \\
\vdots \\
b_m 1x1 + bm2x_2 + \cdots + bmnx_n \leq c_m \\
x_i \in \mathbb{N} \text{ o } x_i \in [0,1] \subset \mathbb{N}
\end{gathered}
$$

Aplicaciones (reales) de los Problemas

Muchos de estos problemas existen en la practica:

- Las empresas de transporte como Fedex, UPS y DHL se enfrentan constantemente al problema de la mochila,
- Universidades y centros educacionales se enfrentan constantemente al problema de planificacion de salas y horarios, el cual generalmente se reduce al coloreado de grafos.
- Planificar la ejecucion de tareas en varias CPUs o en un Cluster de computo requiere resolver el problema de planificacion de tareas.
- Muchos problemas financieros o de Ingenieria se resuelven a traves de programacion lineal entera/booleana.

Aplicaciones (reales) de los Problemas

- Un cartero o algun servicio de delivery se enfrenta constantemente al problema del vendedor viajero.
- Los vehiculos de Google Street View requieren resolver el problema del Camino Euleriano para poder visitar todas las calles de Santiago.
- El problema de cortar barras u otro elemento de produccion es tambien un caso del problema de la mochila.
- En Robotica y Transporte, crear las rutas de viaje en una ciudad es un problema combinacional complejo, el cual se puede modelar como varios de los problemas vistos.

Preliminares

Teoria de la Complejidad Computacional

Algoritmos Aproximados

Introduccion

Pregunta

Es posible resolver todos los problemas computacionales con algoritmos de tiempo polinomial?

Introduccion

Pregunta

Es posible resolver todos los problemas computacionales con algoritmos de tiempo polinomial?

Respuesta

NO!

Problema de Optimizacion

Los problemas de optimizacion son los problemas clasicos que hemos visto hasta ahora, donde el objetivo es tipicamente maximizar o minimizar una cierta funcion objetivo.

Formalizacion

Formalmente, un problema posee varias soluciones factibles (osea, legales) cada una con un valor asociado. Un problema de optimizacion es obtener la solucion factible con el **mejor valor**[1].

[1] esto considera minimizar o maximizar, dependiendo del caso

Problema de Decision

Los problemas de decision son los problemas que tienen respuesta binaria (verdadero o falso).

Ejemplo

- Es posible colorear un grafo con a lo mas 3 colores?
- Es posible planificar ciertas tareas y cumplir el plazo dentro de las restricciones de recursos?

Cualquier problema de optimizacion se puede plantear como un problema de decision, imponiendo un umbral en el valor a optimizar y convirtiendo el problema a una pregunta.

Ejemplo

Problema de Optimizacion

Minimizar el numero de colores requerido para colorear un grafo.

Problema de Decision

Es posible colorear un grafo con a lo mas K colores?

Problema de Optimizacion

Minimizar el numero de cajones requerido para empacar un cierto conjunto de objetos.

Problema de Decision

Es posible empacar un cierto conjunto de objetos en a lo mas K cajones?

Clase P

Definicion

P es el conjunto de problemas de decision que poseen solucion de tiempo polinomial, en una maquina determinista.

En palabras simples

P es el conjunto de problemas que tienen solucion con complejidad algoritmica del orden $O(n^p)$.

Clase NP

Definicion

- NP es el conjunto de problemas de decision cuyas soluciones pueden ser verificadas en tiempo polinomial.
- Esta definicion es equivalente al conjunto de problemas que poseen solucion en tiempo polinomial, en una maquina no determinista.

En palabras simples

- NP es el conjunto de problemas que tienen solucion con complejidad algoritmica que no es de orden polinomial ($\notin O(n^p)$).
- NP es el conjunto de problemas que tienen solucion con complejidad algoritmica de orden $O(e^n)$ o $(n!)$.

Como funciona NP?

Un problema que pertenece a NP puede ser resuelto por un algoritmo no-deterministico, que funciona asi:

1. "Adivinar" la solucion al problema.
2. Verificar la solucion adivinada en el paso anterior.

Adivinar una solucion tiene un costo $O(1)$ en una maquina no deterministica, y verificar la solucion tiene un costo polinomial $O(n^p)$, por ende el costo completo del algoritmo es $O(n^p)$. El problema de esta aproximacion es que no existen maquinas no deterministicas que puedan adivinar soluciones.

Ejemplos de Problemas en P y NP

P

Problemas que hemos visto en el curso, como Ordenamiento, Seleccion, Busqueda, muchos algoritmos de Programacion Dinamica, etc.

NP

SAT, Coloreado de grafos, Planificacion de Tareas, Problema de la Mochila con algunas restricciones, el Problema del Vendedor Viajero, Ciclos Hamiltonianos, Caminos Eulerianos, Programacion Lineal Entera/Booleana, etc.

Reduccion de Problemas

Imaginemos que tenemos 2 problemas, P y Q, pero tenemos un algoritmo solo para resolver Q. Supongamos que tenemos una funcion T que posee la siguiente propiedad:

T toma una entrada x para P, y produce T(x), una entrada para Q, tal que la respuesta correcta para P con x es "si" si y solo si la respuesta de Q con T(x) es tambien "si".

Entonces, si construimos T y tenemos el algoritmo para resolver Q, tenemos tambien un algoitmo para resolver P. Esto se llama reducir un problema P a un problema Q.

Reduccion de Problemas

Si calcular T toma tiempo $O(n^p)$, entonces la reduccion se denomina reduccion en tiempo polinomial.

- La reduccion indica que un problema P es al menos tan dificil de resolver como un problema Q.
- Por ende, el algoritmo que soluciona P tiene la misma complejidad que la solucion de Q, a menos que la complejidad de la reduccion sea mayor.

Ejemplo

Es posible reducir un problema de planificacion de horarios de clases y sus correspondientes salas a un problema de coloreado de grafos:

- Se crea un grafo en el cual cada nodo identifica una clase en un cierto horario.
- Se conectan los nodos A y B con una arista si y solo si las clases A y B se dictan en el mismo horario, y por ende, requieren diferente sala de clases.
- Se colorea el grafo, donde color representa una sala de clases diferente asignada a dicha clase en dicho horario.
- El numero minimo de colores requerido para colorear el grafo corresponde al minimo numero de salas de clase requeridas para dictar todas las asignaturas.
- La complejidad de la reduccion es $O(n^2)$, por lo cual es una reduccion polinomial.

Problemas NP-completos

Definicion

Un problema L es NP-completo si:

- $L \in NP$
- Cada problema $C \in NP$ es reducible a L en tiempo polinomial.

Problemas NP-completos

SAT fue el primer problema que se demostro que es NP-completo. Desde ese problema se han reducido muchos problemas a SAT, demostrando que son NP-completos, entre ellos:

- Problema del Vendedor Viajero, en su version de problema de decision.
- Problema de la Mochila.
- Coloreado de Grafos.

Problemas NP-duro

Definicion

- $L \notin NP$
- Cada problema $C \in NP$ es reducible a L en tiempo polinomial.

Los problemas NP-duro son problemas al menos tan dificiles como los problemas NP mas dificiles. Pueden no ser problemas de decision.

Problemas NP-duro

- El problema del vendedor viajero, en su version de problema de optimizacion.
- Problema de Suma de Subconjuntos: Es posible dividir un conjunto en 2 subconjuntos, tal que la suma de ambos subconjuntos sea igual?
- El problema de la parada.

El problema de la parada

Este problema esta planteado de la siguiente forma:

Definicion

Dado un programa computacional, y un valor de la entrada de dicho programa, decidir si el programa termina su ejecucion en algun momento, o se ejecuta infinitamente.

Este problema no tiene solucion conocida, y adicionalmente, es un problema NP-duro.

Propiedades

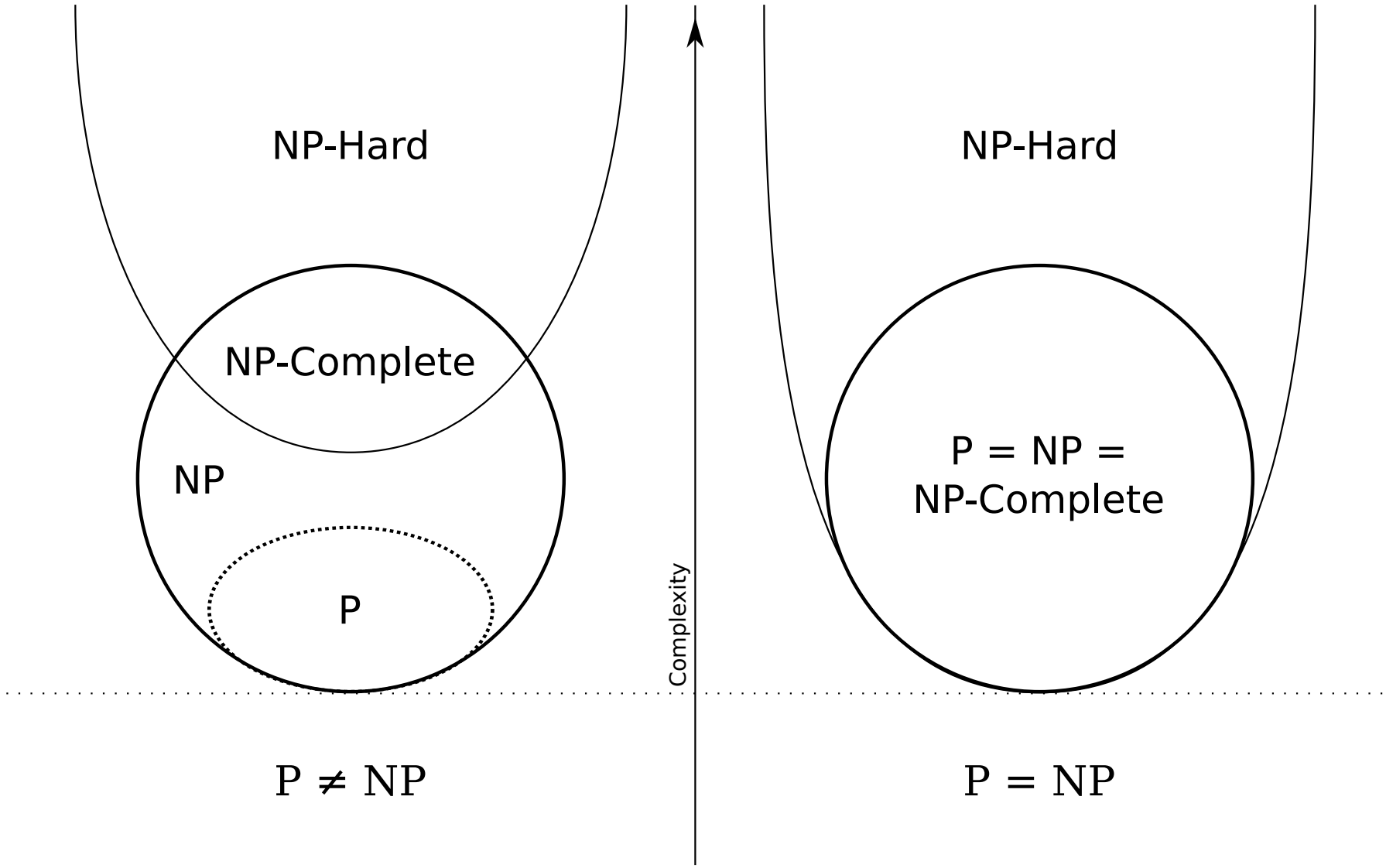
Es bastante obvio que:

$$P \subseteq NP$$

Pero adicionalmente, si se resuelve cualquier problema NP-completo en tiempo polinomial, esto implicaria:

$$P = NP$$

Como no existen (hasta ahora) algoritmos polinomiales para resolver problemas NP-completos, la relacion de igualdad entre P y NP sigue siendo una duda.


NP-Hard
NP-Complete
NP
P
Complexity
NP-Hard
P = NP =
NP-Complete
P ≠ NP
P = NP

Problema Fundamental de la Ciencia de la Computacion

Dado que $P \subseteq NP$, es $P = NP$?

Esta interrogante no tiene respuesta en la actualidad, y su respuesta tanto positiva como negativa tiene profundas implicancias en la Ciencia de la Computacion.

Otras Clases de Complejidad

Existen otras clases de complejidad:

PSPACE

Conjunto de todos los problemas de decision que pueden ser resueltos usando una cantidad polinomial de espacio. Esto implica que la complejidad espacial de dichos algoritmos esta en $O(n^p)$.

EXPTIME

Conjunto de todos los problemas de decision que poseen solucion de tiempo exponencial $O(e^n)$ en una maquina deterministica.

EXPSPACE

Conjunto de todos los problemas de decision que pueden ser resueltos usando una cantidad exponencial de espacio. Esto implica que la complejidad espacial de dichos algoritmos esta en $O(e^n)$.

Otras Clases de Complejidad

$$P \subseteq NP \subseteq PSPACE$$

$$PSPACE \subseteq EXPTIME \subseteq EXPSPACE$$

Porque es importante todo esto?

Dada esta teoria, la pregunta que les debe dominar la mente en este momento es:

Para que sirve toda la teoria de P y NP?

Porque es importante todo esto?

Dada esta teoria, la pregunta que les debe dominar la mente en este momento es:

Para que sirve toda la teoria de P y NP?

La respuesta es que no tiene alguna aplicacion practica para resolver problemas. Sin embargo, existe un "pero".

Porque es importante todo esto?

La teoria de la complejidad algoritmica les dice varias cosas:

- No todos los problemas son igual de faciles/dificiles de resolver.
- Existen 2 clases de problemas, los faciles (P), y los dificiles (NP).
- Para los problemas dificiles, no existen algoritmos que los resuelvan en tiempo polinomico, pero existen algoritmos que lo hacen en tiempo $O(e^n)$ o $O(n!)$, lo cual en la practica no es factible.
- Existen algoritmos aproximados que resuelven problemas NP de forma aproximada.

Porque es importante todo esto?

- En la practica, los problemas que estan en NP son importantes, y mucha investigacion se dedica a encontrar soluciones polinomicas a ellos.
- Por ende, cuando se enfrenten a un problema que esta en NP durante su ejercicio profesional, saben que es un problema dificil, que no poseen solucion directa, y que solo se pueden usar soluciones aproximadas (tipicamente son heuristicas).
- Asi que en el ejercicio profesional, no prometan resolver problemas que esta en NP de forma exacta.

Preliminares

Teoria de la Complejidad Computacional

Algoritmos Aproximados

Motivacion

Muchos problemas son NP o NP-completo, por ende no existen en la actualidad algoritmos eficientes para resolverlos.

- Para valores pequeños de la entrada, un algoritmo de complejidad exponencial puede ser adecuado.
- Aislar casos especiales, los cuales pueden ser resueltos de forma mas eficiente.
- Relajar las restricciones y buscar soluciones cercanas al optimo (aproximadas).

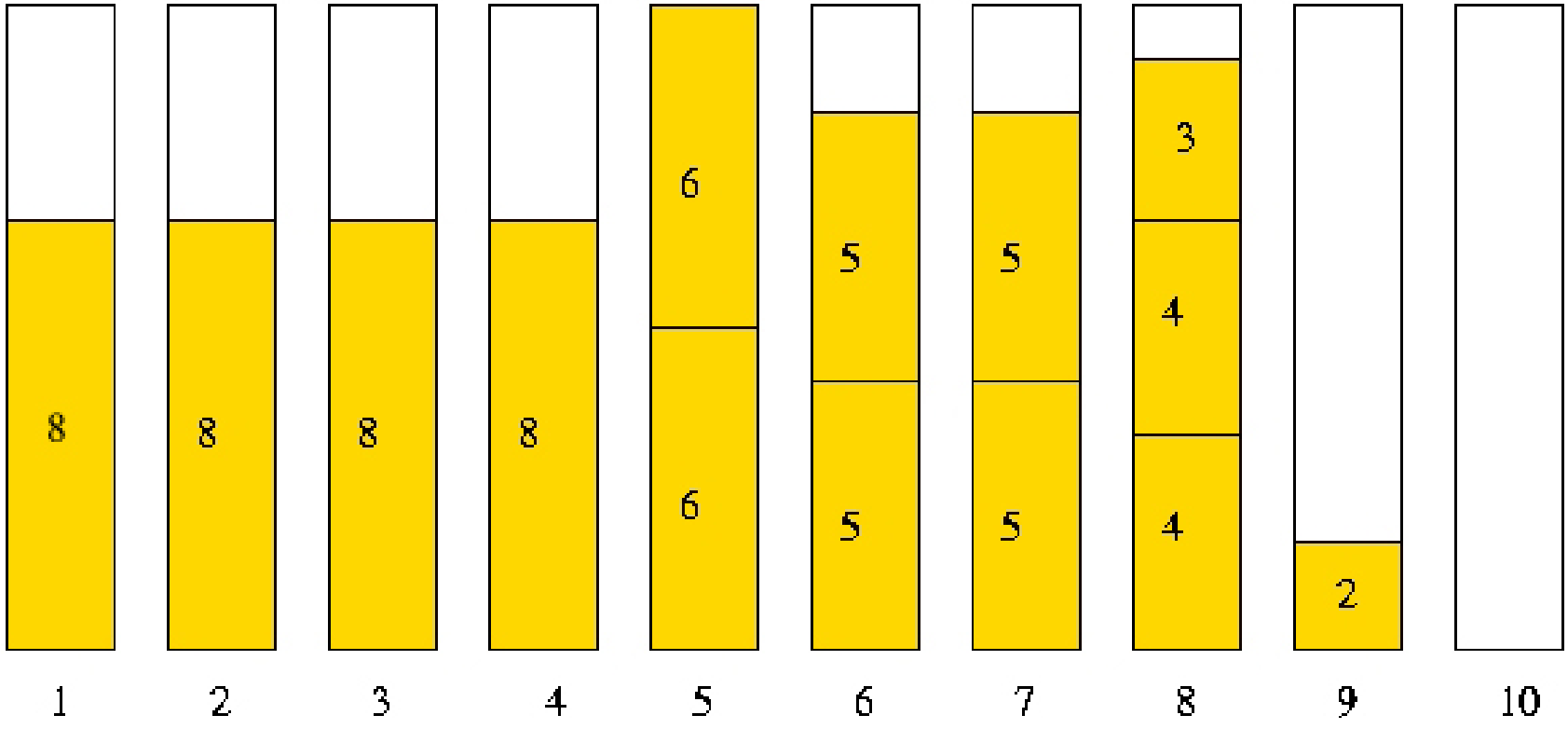
Empaquetado de Cajones

Empacado

Supongamos que tenemos un numero ilimitado de cajones de capacidad unitaria, y objetos de tamaño s_i, donde los $s_i \in [0, 1]$. Se debe encontrar un empacado de los objetos que minimice el numero de cajones usados.

Primer Ajuste Decreciente

Consiste en ordenar los objetos en orden decreciente por tamaño, y ir asignando cada objeto desde el mas grande al mas pequeño, uno a uno.


8
8
8
8
6
6
5
5
5
5
4
4
3
2
1
2
3
4
5
6
7
8
9
10

Primer Ajuste Decreciente

La complejidad de dicho algoritmo es $\Theta(n^2)$ en el peor caso, por lo que es una solucion polinomial aproximada. Si denotamos por opt(S) el valor optimo real (el cual no conocemos), el algoritmo presenta las siguientes propiedades:

- Todos los objetos colocados en cajones extra (cajones con indice mayor a opt(S)) tienen un uso menor a $\frac{1}{3}$
- El numero de objetos que FFD coloca en cajones extra es a lo mas opt(S) - 1.

Coloreado de Grafos

Coloreado secuencial

```
void colorSec(V, E)
{
    for(int i = 1; i <= n; i++) {
        for(int c = 1; c <= n; c++) {
            Si ningun vertice adyacente a V[i]
            tiene el color c.
                Colorear V[i] con c.
                break;
        }
    }
}
```

Coloreado de Grafos

Coloreado aproximado

```
void colorAprox(G)
{
    int k = 1;
    bool coloreado = false;

    while(!coloreado) {
        k = 2 * k;
        coloreado = color(G, k);
    }
}
```

Luego de ejecutarse el algoritmo, el valor optimo de colores esta entre k y $\frac{k}{2}$, por lo cual se puede ejecutar una busqueda binaria para encontrar el numero optimo de colores. Este algoritmo requiere otro algoritmo que coloree un grafo con a lo mas k colores.

Problema del Vendedor Viajero

Vecino mas Cercano

```
void masCercanoTSP(V, E, W)
{
    Seleccionar un vertice arbitrario
    para iniciar el ciclo C.
    v = s;
    Mientras hayan vertices que aun no esten en C:
        Seleccionar una arista vw de peso minimo,
        donde w no esta en C.
        Agregar la arista vw a C.
        v = w;

    Agregar la arista vs a C;
    retornar C.
}
```

Problema del Vendedor Viajero

Eslabon mas Corto

```
void eslabonMasCortoTSP(V, E, W)
{
    R = E;       // Aristas restantes
    C = Vacio;   // Aristas del ciclo.
    Mientras R no este vacio:
        Quitar la arista mas corta vw de R.
        Si vw no forma un ciclo con las aristas de C,
        y vw no seria la tercera arista de C
        que incide en v o w:
            Agregar vw a C.

    Agregar la arista que conecta
    los extremos del camino que esta en C.
    return C;
}
```

Bibliografia

Para esta unidad, se recomienda leer:

1. Capitulo 13 del Libro “Algoritmos Computacionales: Introduccion al Analisis y Diseño” de Sara Baase y Allen Van Gelder.