Rappel : La taille de la donnée

Taille de la donnée $|x|$

- Espace mémoire nécessaire pour la représenter x
- Dans une unité quelconque, fonction de la donnée : $|x| \in \Theta(f(x))$

Rappel : La taille de la donnée

Taille de la donnée $|x|$

- Espace mémoire nécessaire pour la représenter x
- Dans une unité quelconque, fonction de la donnée : $|x| \in \Theta(f(x))$

- Type primitif entier, flottant, etc. sur 32 ou 64 bits : $\Theta(1)$
- Entier non-borné x : $|x| \in \Theta(\log x)$
- Liste L de n caractères : $|L| \in \Theta(n)$

La taille de la donnée : exemple

- La complexité se mesure **en fonction de la taille de la donnée**
- Parfois, on mesure plus facilement la complexité d'un algorithme en fonction d'un autre paramètre que la taille de la donnée (ex : sa **valeur**)
- Si l'algorithme **A** est en $\Theta(f(x))$ pour une donnée x et la taille $|x|$ est en $\Theta(g(x))$, alors la complexité de **A** sera en $\Theta(f(g^{-1}(x)))$

La taille de la donnée : exemple

- La complexité se mesure **en fonction de la taille de la donnée**
- Parfois, on mesure plus facilement la complexité d'un algorithme en fonction d'un autre paramètre que la taille de la donnée (ex : sa **valeur**)
- Si l'algorithme **A** est en $\Theta(f(x))$ pour une donnée x et la taille $|x|$ est en $\Theta(g(x))$, alors la complexité de **A** sera en $\Theta(f(g^{-1}(x)))$

Exemple

Algorithme : Carré(x)

Données : un entier x
Résultat : un entier valant x^2
r : entier;
début
    $r \leftarrow 0$;
    pour *i allant de* 1 *à* x **faire**
        pour *j allant de* 1 *à* x **faire**
            $r \leftarrow r + 1$;
    retourner r;

La taille de la donnée : exemple

- La complexité se mesure **en fonction de la taille de la donnée**
- Parfois, on mesure plus facilement la complexité d'un algorithme en fonction d'un autre paramètre que la taille de la donnée (ex : sa **valeur**)
- Si l'algorithme **A** est en $\Theta(f(x))$ pour une donnée x et la taille $|x|$ est en $\Theta(g(x))$, alors la complexité de **A** sera en $\Theta(f(g^{-1}(x)))$

Exemple

Algorithme : Carré(x)

Données : un entier x
Résultat : un entier valant x^2

```
r : entier;
début
    r ← 0;
    pour i allant de 1 à x faire
        pour j allant de 1 à x faire
            r ← r + 1;
    retourner r;
```

- Complexité : $\Theta(x^2)$

La taille de la donnée : exemple

- La complexité se mesure **en fonction de la taille de la donnée**
- Parfois, on mesure plus facilement la complexité d'un algorithme en fonction d'un autre paramètre que la taille de la donnée (ex : sa **valeur**)
- Si l'algorithme **A** est en $\Theta(f(x))$ pour une donnée x et la taille $|x|$ est en $\Theta(g(x))$, alors la complexité de **A** sera en $\Theta(f(g^{-1}(x)))$

Exemple

Algorithme : Carré(x)

Données : un entier x
Résultat : un entier valant x^2

```
r : entier;
début
    r ← 0;
    pour i allant de 1 à x faire
        pour j allant de 1 à x faire
            r ← r + 1;
    retourner r;
```

- Complexité : $\Theta(x^2)$
- Mais la taille de la donnée est $|x| = \Theta(\log x)$

La taille de la donnée : exemple

- La complexité se mesure **en fonction de la taille de la donnée**
- Parfois, on mesure plus facilement la complexité d'un algorithme en fonction d'un autre paramètre que la taille de la donnée (ex : sa **valeur**)
- Si l'algorithme **A** est en $\Theta(f(x))$ pour une donnée x et la taille $|x|$ est en $\Theta(g(x))$, alors la complexité de **A** sera en $\Theta(f(g^{-1}(x)))$

Exemple

Algorithme : Carré(x)

Données : un entier x
Résultat : un entier valant x^2

```
r : entier;
début
    r ← 0;
    pour i allant de 1 à x faire
        pour j allant de 1 à x faire
            r ← r + 1;
    retourner r;
```

- Complexité : $\Theta(x^2)$
- Mais la taille de la donnée est $|x| = \Theta(\log x)$
- Autrement dit, $x = \Theta(2^{|x|})$

La taille de la donnée : exemple

- La complexité se mesure **en fonction de la taille de la donnée**
- Parfois, on mesure plus facilement la complexité d'un algorithme en fonction d'un autre paramètre que la taille de la donnée (ex : sa **valeur**)
- Si l'algorithme **A** est en $\Theta(f(x))$ pour une donnée x et la taille $|x|$ est en $\Theta(g(x))$, alors la complexité de **A** sera en $\Theta(f(g^{-1}(x)))$

Exemple

Algorithme : Carré(x)

Données : un entier x
Résultat : un entier valant x^2

```
r : entier;
début
    r ← 0;
    pour i allant de 1 à x faire
        pour j allant de 1 à x faire
            r ← r + 1;
    retourner r;
```

- Complexité : $\Theta(x^2)$
- Mais la taille de la donnée est $|x| = \Theta(\log x)$
- Autrement dit, $x = \Theta(2^{|x|})$
- cet algorithme est donc en $\Theta(2^{2|x|})$ (exponentiel !)

Rappel : La taille d'une structures de données

- Dans le cas de structures de données (listes, ensembles, etc.) il faut analyser la structure : la taille de chaque élément, les pointeurs, etc.

Rappel : La taille d'une structures de données

- Dans le cas de structures de données (listes, ensembles, etc.) il faut analyser la structure : la taille de chaque élément, les pointeurs, etc.

- La taille des structures va dépendre de leur codage :

Rappel : La taille d'une structures de données

- Dans le cas de structures de données (listes, ensembles, etc.) il faut analyser la structure : la taille de chaque élément, les pointeurs, etc.

- La taille des structures va dépendre de leur codage :

Exemple, liste L de n éléments

Rappel : La taille d'une structures de données

- Dans le cas de structures de données (listes, ensembles, etc.) il faut analyser la structure : la taille de chaque élément, les pointeurs, etc.

- La taille des structures va dépendre de leur codage :

Exemple, liste L de n éléments

Il faut $\Theta(1)$ bits par élément, plus la somme des tailles des éléments : donc $|L| \in \Theta(n) + \sum_{e \in} |e| \subseteq \Theta(n) \times \Theta(|e|)$

Rappel : La taille d'une structures de données

- Dans le cas de structures de données (listes, ensembles, etc.) il faut analyser la structure : la taille de chaque élément, les pointeurs, etc.

- La taille des structures va dépendre de leur codage :

Exemple, liste L de n éléments

Il faut $\Theta(1)$ bits par élément, plus la somme des tailles des éléments : donc $|L| \in \Theta(n) + \sum_{e \in} |e| \subseteq \Theta(n) \times \Theta(|e|)$

- Les éléments sont des entiers : $|L| \in \Theta(n)$
- Les éléments sont des entiers non bornés et m est l'élément maximum : $|L| \in \Theta(n \log m)$

Classe “Temps Polynômial”

- Rappel :

TIME $(f(n))$

Ensemble des problèmes pour lesquels il existe un algorithme en $\mathcal{O}(f(n))$

Classe "Temps Polynômial"

- Rappel :

TIME $(f(n))$

Ensemble des problèmes pour lesquels il existe un algorithme en $\mathcal{O}(f(n))$

- Classe des problèmes pour lesquels il existe un algorithme polynômial :

PTIME ou simplement : P

Ensemble des problèmes pour lesquels il existe un algorithme en $\mathcal{O}(n^c)$ pour une constante c.

$$\mathbf{P} = \bigcup_{c \geq 0} \mathbf{TIME}(n^c)$$

Les Classes P et NP

Classe “Temps Polynômial”

- Rappel :

TIME $(f(n))$

Ensemble des problèmes pour lesquels il existe un algorithme en $\mathcal{O}(f(n))$

Classe "Temps Polynômial"

- Rappel :

TIME $(f(n))$

Ensemble des problèmes pour lesquels il existe un algorithme en $\mathcal{O}(f(n))$

- Classe des problèmes pour lesquels il existe un algorithme polynômial :

PTIME ou simplement : P

Ensemble des problèmes pour lesquels il existe un algorithme en $\mathcal{O}(n^c)$ pour une constante c.

$$\mathbf{P} = \bigcup_{c \geq 0} \mathbf{TIME}(n^c)$$

Classe “Temps Exponentiel”

- Classe des problèmes pour lesquels il existe un algorithme exponentiel :

Classe "Temps Exponentiel"

- Classe des problèmes pour lesquels il existe un algorithme exponentiel :

EXPTIME :

Ensemble des problèmes pour lesquels il existe un algorithme en $\mathcal{O}(2^{n^c})$ pour une constante c

$$\mathbf{EXPTIME} = \bigcup_{c \geq 1} \mathbf{TIME}(2^{n^c})$$

Classe "Temps Exponentiel"

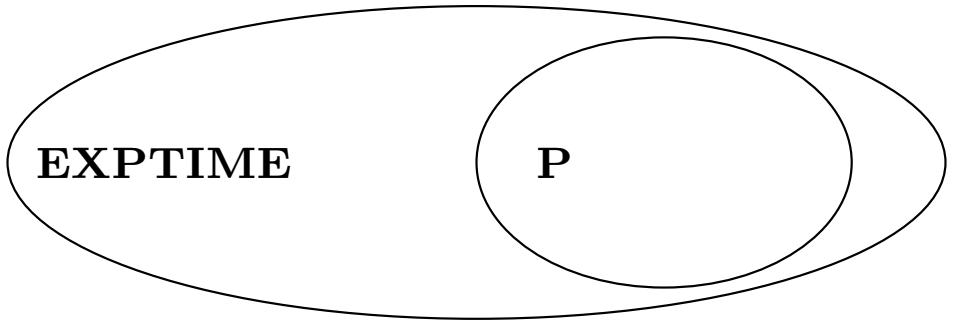
- Classe des problèmes pour lesquels il existe un algorithme exponentiel :

EXPTIME :

Ensemble des problèmes pour lesquels il existe un algorithme en $\mathcal{O}(2^{n^c})$ pour une constante c

$$\mathbf{EXPTIME} = \bigcup_{c \geq 1} \mathbf{TIME}(2^{n^c})$$

- Evidemment, $\mathbf{P} \subseteq \mathbf{EXPTIME}$



Classe "Temps Exponentiel"

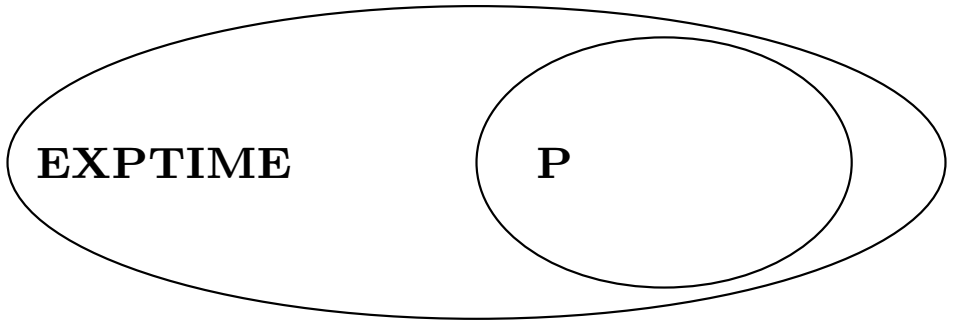
- Classe des problèmes pour lesquels il existe un algorithme exponentiel :

EXPTIME :

Ensemble des problèmes pour lesquels il existe un algorithme en $\mathcal{O}(2^{n^c})$ pour une constante c

$$\mathbf{EXPTIME} = \bigcup_{c \geq 1} \mathbf{TIME}(2^{n^c})$$

- Evidemment, $\mathbf{P} \subseteq \mathbf{EXPTIME}$
- Est-ce que $\mathbf{P} \subset \mathbf{EXPTIME}$?



Classe "Temps Exponentiel"

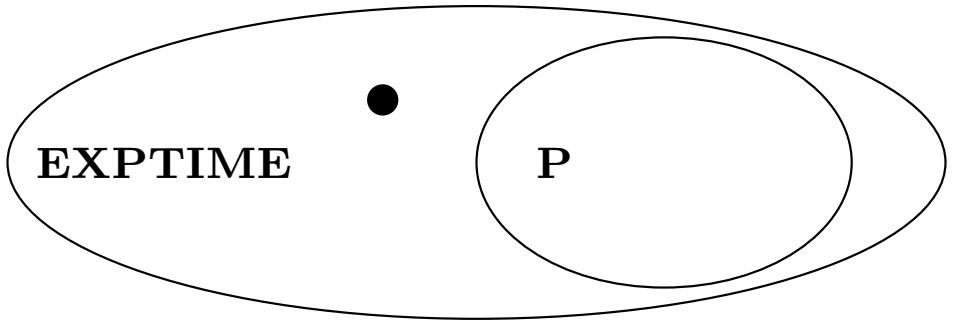
- Classe des problèmes pour lesquels il existe un algorithme exponentiel :

EXPTIME :

Ensemble des problèmes pour lesquels il existe un algorithme en $\mathcal{O}(2^{n^c})$ pour une constante c

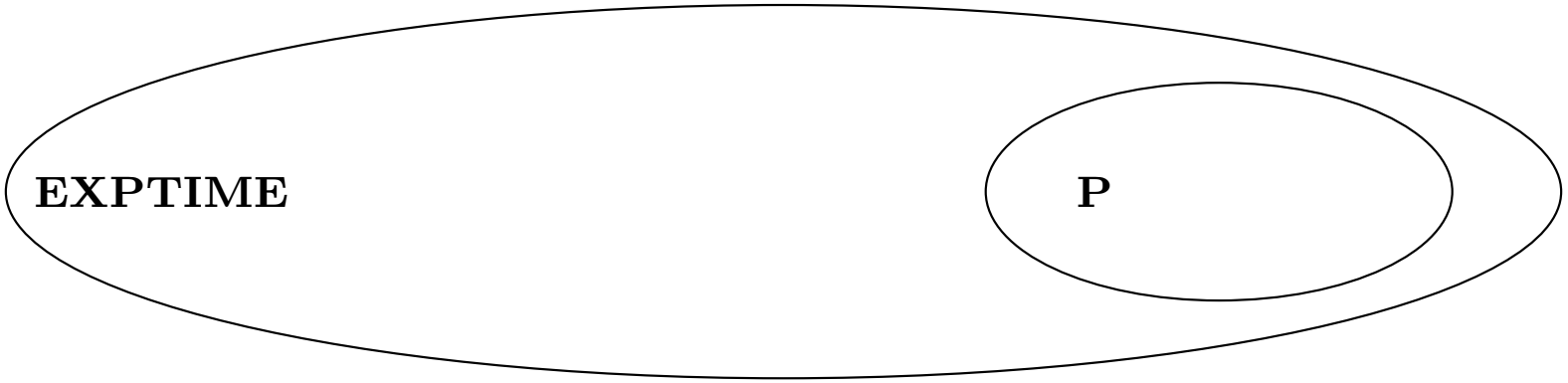
$$\mathbf{EXPTIME} = \bigcup_{c \geq 1} \mathbf{TIME}(2^{n^c})$$

- Evidemment, $\mathbf{P} \subseteq \mathbf{EXPTIME}$
- Est-ce que $\mathbf{P} \subset \mathbf{EXPTIME}$? **Oui !**
- Il existe des problèmes en $\Omega(2^n)$



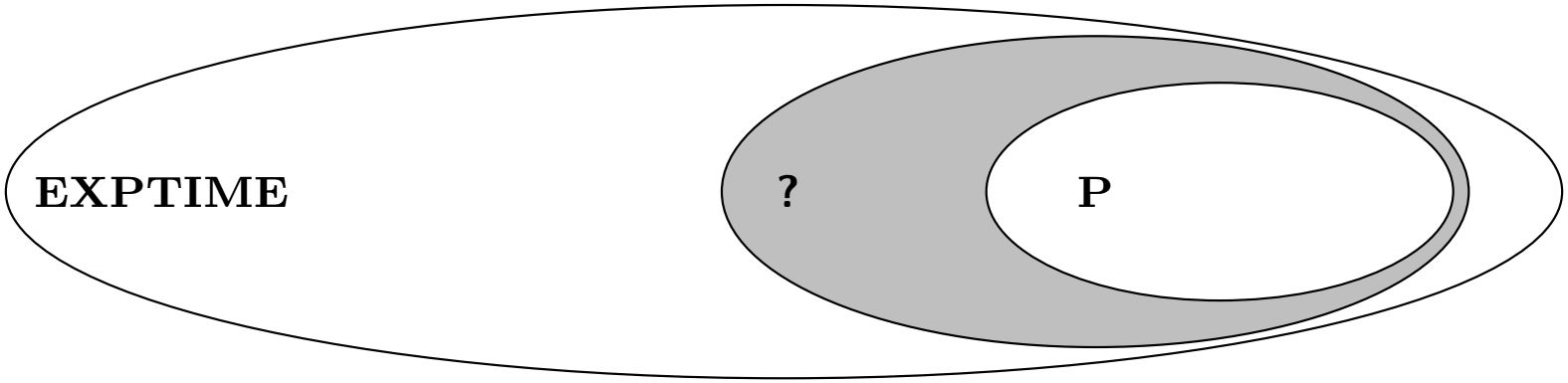
Problèmes “intermédiaires”

- Une classification trop grossière ?



Problèmes “intermédiaires”

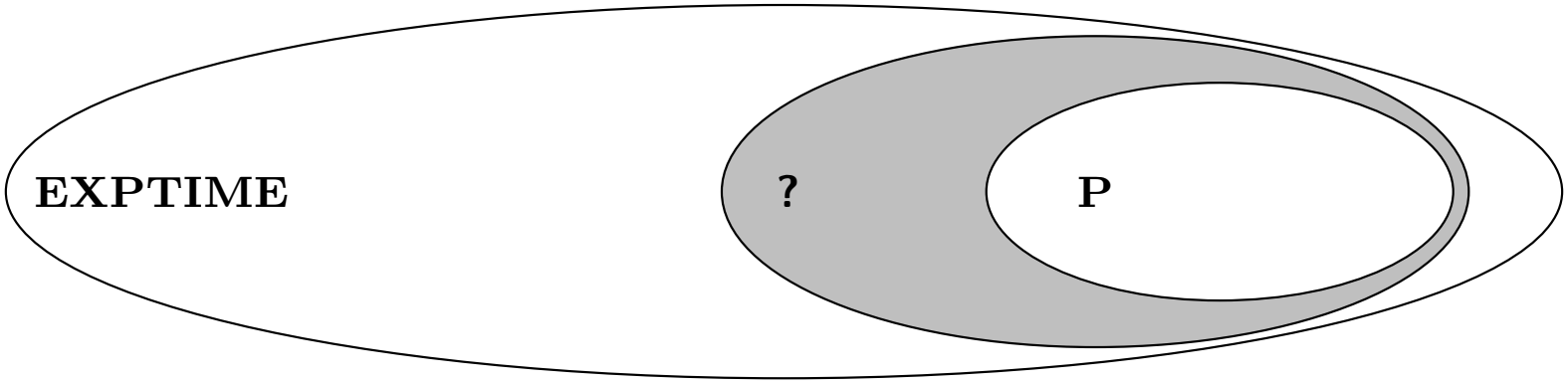
- Une classification trop grossière ?



- Il existe des problèmes pour lesquels on ne connait pas d’algorithme en temps polynômial, sans pouvoir prouver qu’ils n’en n’ont pas

Problèmes “intermédiaires”

- Une classification trop grossière ?



- Il existe des problèmes pour lesquels on ne connait pas d'algorithme en temps polynômial, sans pouvoir prouver qu'ils n'en n'ont pas
- Ces problèmes sont très nombreux...
- ... et très intéressants : *Voyageur de Commerce*, *Programmation Linéaire en Nombres Entiers*, *SAT*, *Isomorphisme de Graphes*, etc.

Classes non-déterministes

- Puisque on ne peut pas prouver que ces problèmes n'ont pas d'algorithme en temps polynômial, on n'est pas sûr qu'une telle classe **existe**

Classes non-déterministes

- Puisque on ne peut pas prouver que ces problèmes n'ont pas d'algorithme en temps polynômial, on n'est pas sûr qu'une telle classe **existe**
- Si elle existe, alors cette classe correspond à la notion de **non-déterminisme**

Classes non-déterministes

- Puisque on ne peut pas prouver que ces problèmes n'ont pas d'algorithme en temps polynômial, on n'est pas sûr qu'une telle classe **existe**
- Si elle existe, alors cette classe correspond à la notion de **non-déterminisme**

- Rappel de la définition d'un algorithme :

Algorithme : méthode pour résoudre un problème

- Non-ambiguë, composée d'instructions primitives

Classes non-déterministes

- Puisque on ne peut pas prouver que ces problèmes n'ont pas d'algorithme en temps polynômial, on n'est pas sûr qu'une telle classe **existe**
- Si elle existe, alors cette classe correspond à la notion de **non-déterminisme**

- Rappel de la définition d'un algorithme :

Algorithme : méthode pour résoudre un problème

- **Non-ambiguë**, composée d'instructions primitives
- Non-ambiguité = déterminisme

Algorithmes déterministes

Pour un certain **état de la mémoire**, et une certaine **position** dans le code, il n'existe qu'**une seule** instruction possible avec **un seul** effet possible (sur l'état de la mémoire et la position).

Algorithmes déterministes

Pour un certain **état de la mémoire**, et une certaine **position** dans le code, il n'existe qu'**une seule** instruction possible avec **un seul** effet possible (sur l'état de la mémoire et la position).

- **État** : contenu de la mémoire + position d'un *curseur* dans le code

Algorithmes déterministes

Pour un certain **état de la mémoire**, et une certaine **position** dans le code, il n'existe qu'**une seule** instruction possible avec **un seul** effet possible (sur l'état de la mémoire et la position).

- **État** : contenu de la mémoire + position d'un *curseur* dans le code
- **Transition** : Instruction

Algorithmes déterministes

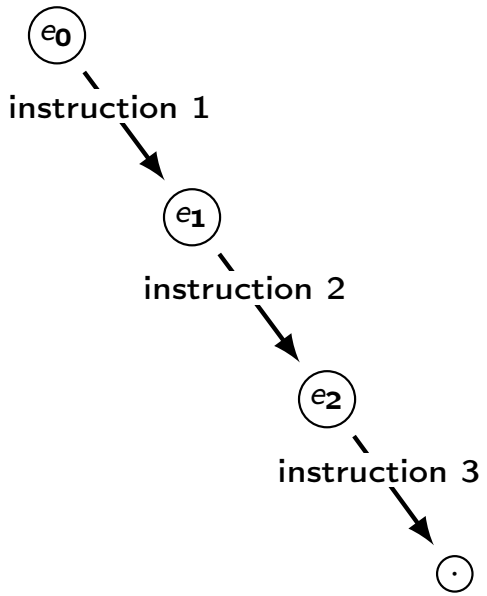
Pour un certain **état de la mémoire**, et une certaine **position** dans le code, il n'existe qu'**une seule** instruction possible avec **un seul** effet possible (sur l'état de la mémoire et la position).

- **État** : contenu de la mémoire + position d'un *curseur* dans le code
- **Transition** : Instruction
- **Effet** : modification de la mémoire et déplacement du curseur

Algorithmes déterministes

Pour un certain **état de la mémoire**, et une certaine **position** dans le code, il n'existe qu'**une seule** instruction possible avec **un seul** effet possible (sur l'état de la mémoire et la position).

- **État** : contenu de la mémoire + position d'un *curseur* dans le code
- **Transition** : Instruction
- **Effet** : modification de la mémoire et déplacement du curseur



Algorithmes déterministes

Pour un certain **état de la mémoire**, et une certaine **position** dans le code, il n'existe qu'**une seule** instruction possible avec **un seule** effet possible (sur l'état de la mémoire et la position).

Algorithme : TriBulles(T)
Données : un tableau T
Résultat : le tableau T trié

```
1 début
2     pour i allant de 2 à |T| faire
3         pour j allant de 1 à i − 1 faire
4             si T[j] > T[j + 1] alors
5                 échanger(T[j], T[j + 1]);
```

Mémoire

T :	5	3	4	9

i : ?

j : ?

Algorithmes déterministes

Pour un certain **état de la mémoire**, et une certaine **position** dans le code, il n'existe qu'**une seule** instruction possible avec **un seule** effet possible (sur l'état de la mémoire et la position).

Algorithme : TriBulles(T)
Données : un tableau T
Résultat : le tableau T trié

```
1 début
2     pour i allant de 2 à |T| faire
3         pour j allant de 1 à i − 1 faire
4             si T[j] > T[j + 1] alors
5                 échanger(T[j], T[j + 1]);
```

Mémoire

T :	5	3	4	9

i : ?

j : ?

Algorithmes déterministes

Pour un certain **état de la mémoire**, et une certaine **position** dans le code, il n'existe qu'**une seule** instruction possible avec **un seule** effet possible (sur l'état de la mémoire et la position).

Algorithme : TriBulles(T)
Données : un tableau T
Résultat : le tableau T trié

```
1 début
2     pour i allant de 2 à |T| faire
3         pour j allant de 1 à i − 1 faire
4             si T[j] > T[j + 1] alors
5                 échanger(T[j], T[j + 1]);
```

Mémoire

T :	5	3	4	9

i : 2

j : ?

Algorithmes déterministes

Pour un certain **état de la mémoire**, et une certaine **position** dans le code, il n'existe qu'**une seule** instruction possible avec **un seule** effet possible (sur l'état de la mémoire et la position).

Algorithme : TriBulles(T)
Données : un tableau T
Résultat : le tableau T trié

```
1 début
2     pour i allant de 2 à |T| faire
3         pour j allant de 1 à i − 1 faire
4             si T[j] > T[j + 1] alors
5                 échanger(T[j], T[j + 1]);
```

Mémoire

T :	5	3	4	9

i : 2

j : 1

Algorithmes déterministes

Pour un certain **état de la mémoire**, et une certaine **position** dans le code, il n'existe qu'**une seule** instruction possible avec **un seule** effet possible (sur l'état de la mémoire et la position).

```
   Algorithme : TriBulles(T)
   Données : un tableau T
   Résultat : le tableau T trié
1  début
2      pour i allant de 2 à |T| faire
3          pour j allant de 1 à i − 1 faire
4              si T[j] > T[j + 1] alors
5                  échanger(T[j], T[j + 1]);
```

Mémoire

T :	5	3	4	9

i : 2

j : 1

Algorithmes déterministes

Pour un certain **état de la mémoire**, et une certaine **position** dans le code, il n'existe qu'**une seule** instruction possible avec **un seule** effet possible (sur l'état de la mémoire et la position).

Algorithme : TriBulles(T)
Données : un tableau T
Résultat : le tableau T trié

```
1 début
2     pour i allant de 2 à |T| faire
3         pour j allant de 1 à i − 1 faire
4             si T[j] > T[j + 1] alors
5                 échanger(T[j], T[j + 1]);
```

Mémoire

T :	3	5	4	9

i : 2

j : 1

Algorithmes non-déterministes

Pour un certain état de la mémoire, et une certaine position dans le code, il peut y avoir **plusieurs** instructions possibles, avec des effets potentiellement différents (sur l'état de la mémoire et la position).

Algorithmes non-déterministes

Pour un certain état de la mémoire, et une certaine position dans le code, il peut y avoir **plusieurs** instructions possibles, avec des effets potentiellement différents (sur l'état de la mémoire et la position).

Algorithme : TriNonDéterministe(T)

Données : un tableau T
Résultat : le tableau T trié

```
1 début
2     pour i allant de 1 à |T| − 1 faire
3         Devine j ∈ [i, .., |T|] t.q. T[j] est minimal;
4         Échange T[i] et T[j];
5         si i > 0 et T[i − 1] > T[i] alors stop;
```

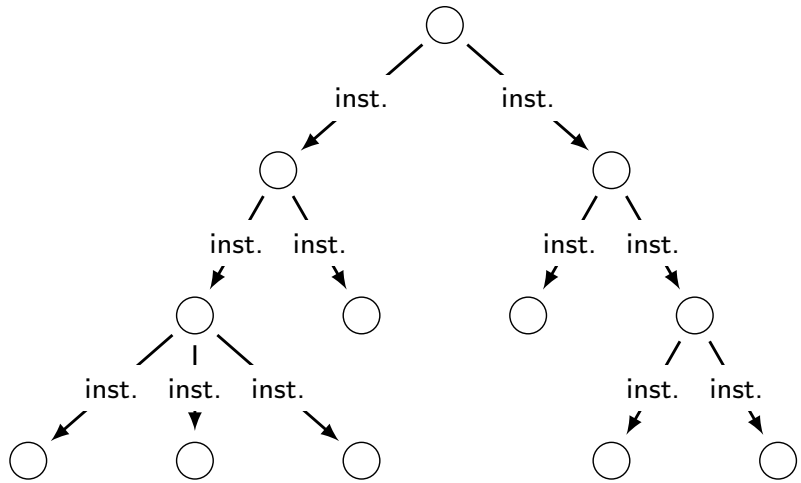
Algorithmes non-déterministes

Pour un certain état de la mémoire, et une certaine position dans le code, il peut y avoir **plusieurs** instructions possibles, avec des effets potentiellement différents (sur l'état de la mémoire et la position).

Algorithme : TriNonDéterministe(T)

Données : un tableau T
Résultat : le tableau T trié

```
1 début
2     pour i allant de 1 à |T| − 1 faire
3         Devine j ∈ [i, .., |T|] t.q. T[j] est minimal;
4         Échange T[i] et T[j];
5         si i > 0 et T[i − 1] > T[i] alors stop;
```



Algorithmes non-déterministes

Algorithme : TriNonDéterministe(T)

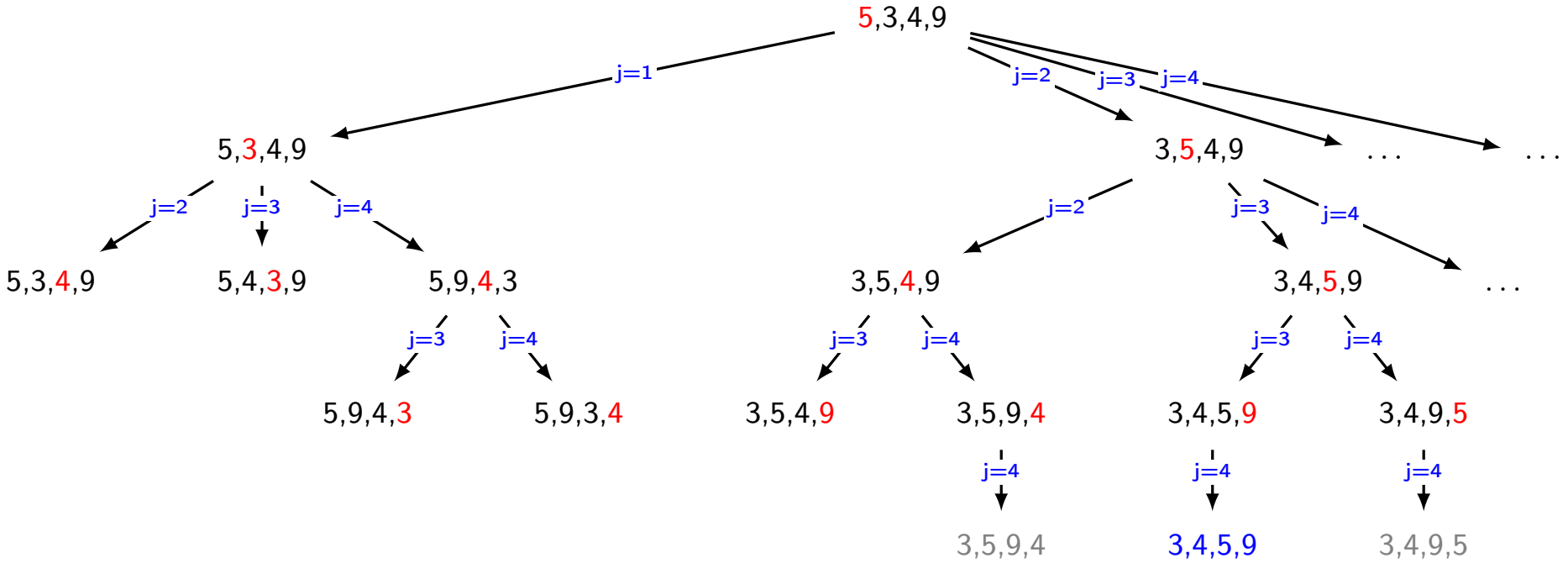
Données : un tableau T
Résultat : le tableau T trié

```
1  début
2      pour i allant de 1 à |T| − 1 faire
3          Devine j ∈ [i, .., |T|] t.q. T[j] est minimal;
4          Échange T[i] et T[j];
5          si i > 0 et T[i − 1] > T[i] alors stop;
```

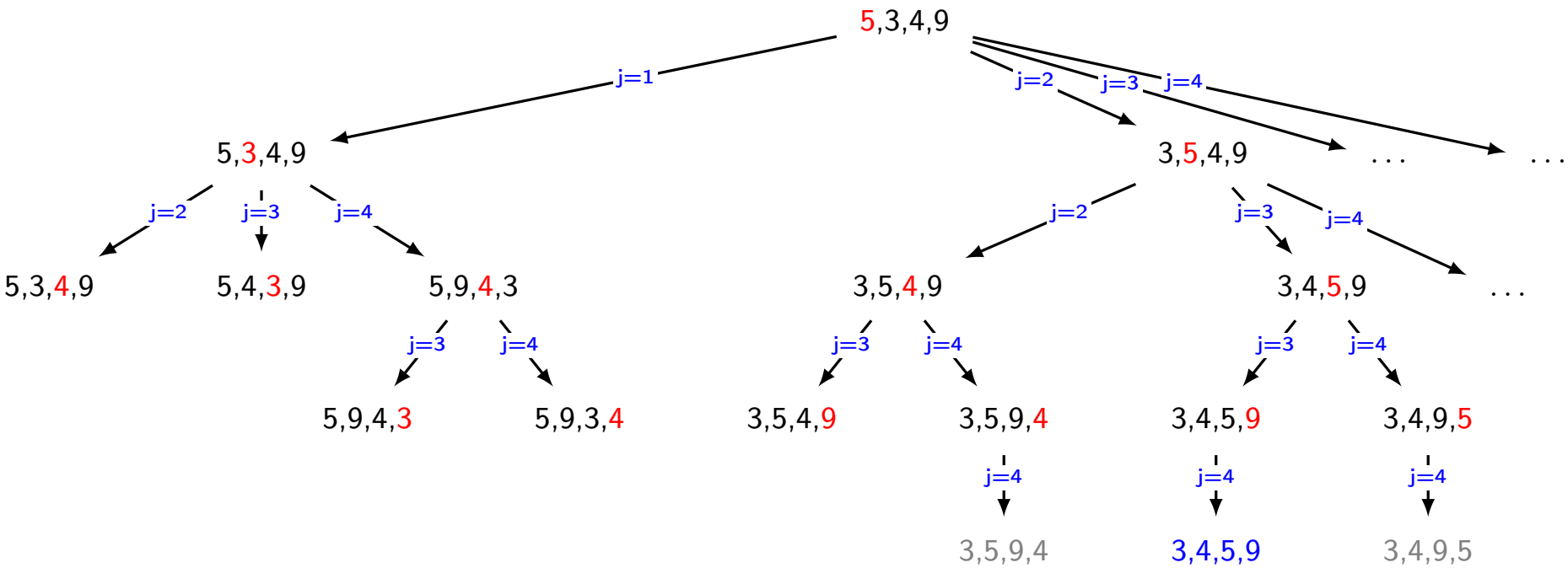
T :

5	3	4	9

i : en rouge

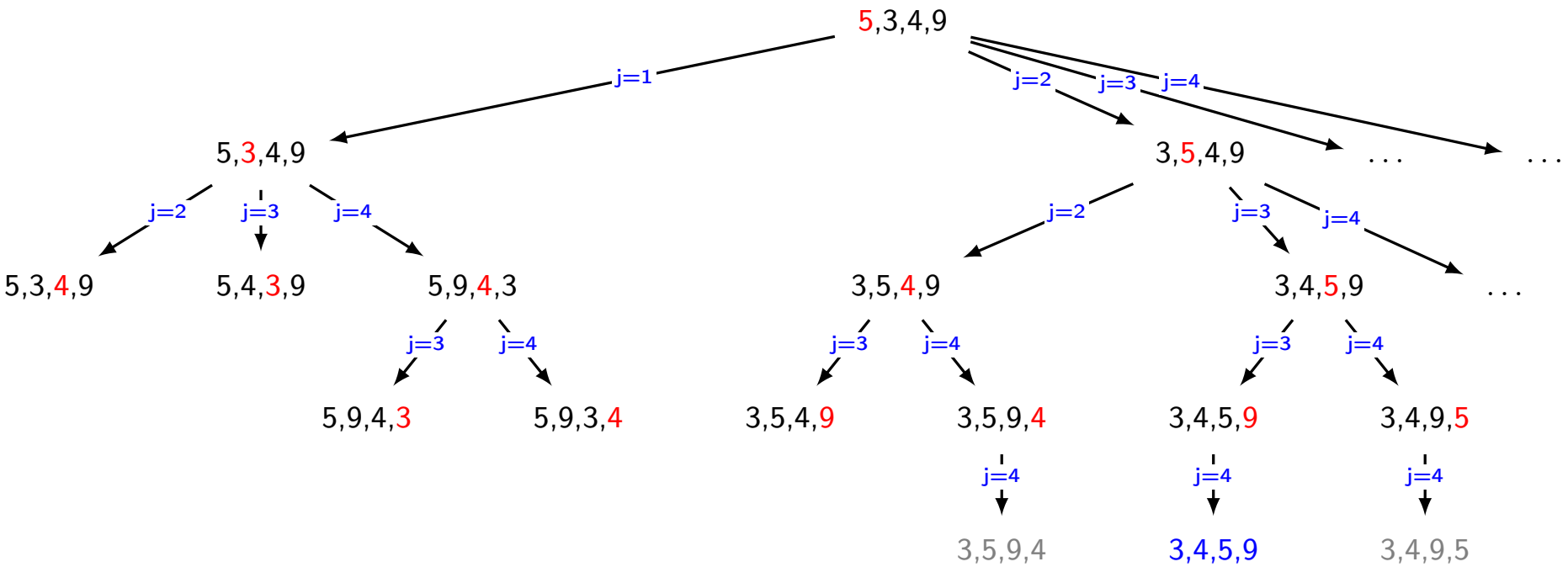


Arbre de calcul non-déterministe



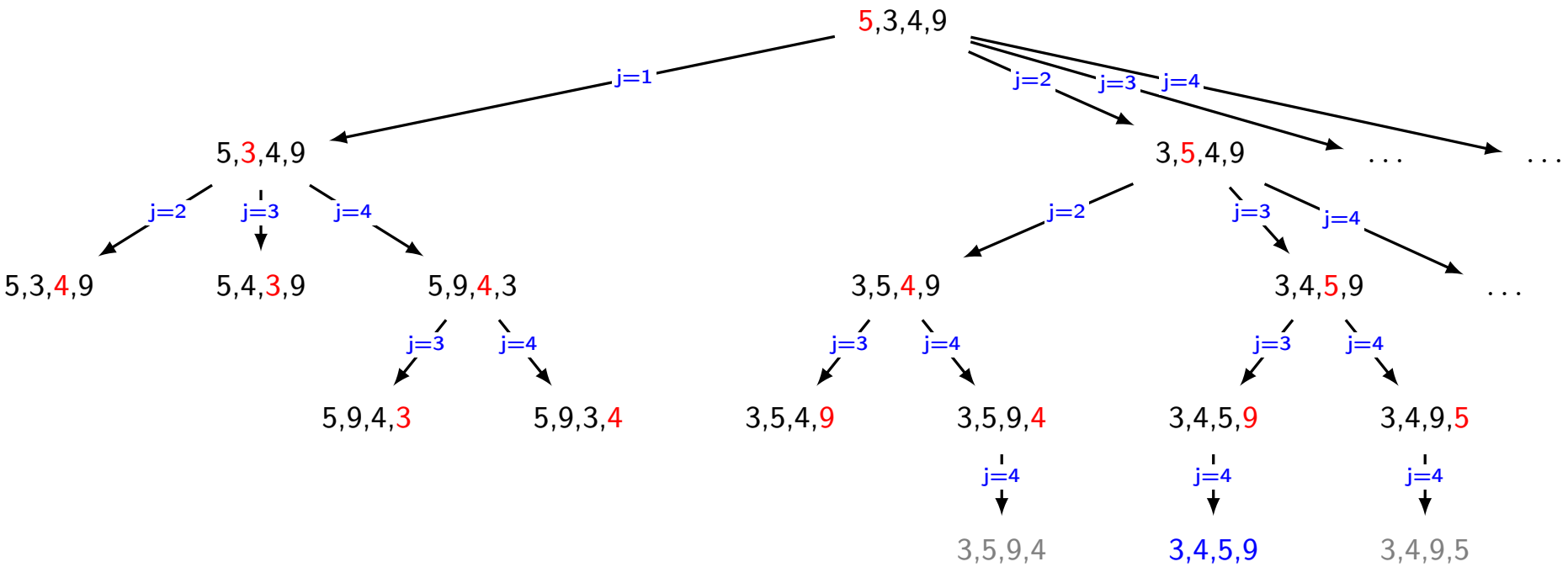
Arbre de calcul non-déterministe

- On suppose que l'algorithme fait **toujours le bon choix** (ou explore tous les choix en parallèle)



Arbre de calcul non-déterministe

- On suppose que l'algorithme fait **toujours le bon choix** (ou explore tous les choix en parallèle)
- Complexité en temps = **longueur maximale d'une branche**
- TriNonDeterministe est en **temps linéaire** ($\mathcal{O}(n)$) puisqu'aucune branche ne peut avoir une longueur supérieure à n



La classe de problèmes NP

- On peut donc définir une classe de complexité en fonction d'une **ressource** (*temps*, *espace*, etc.), une **borne** ($n^c, 2^{n^c}$, etc.) et un **mode** (*déterministe*, *non-déterministe*)
- On se restreint aux problèmes de **décision** : réponse dans $\{\text{oui}, \text{non}\}$

La classe de problèmes NP

- On peut donc définir une classe de complexité en fonction d'une **ressource** (*temps*, *espace*, etc.), une **borne** ($n^c, 2^{n^c}$, etc.) et un **mode** (*déterministe*, *non-déterministe*)
- On se restreint aux problèmes de **décision** : réponse dans $\{\text{oui}, \text{non}\}$

P

Ensemble des problèmes pour lesquels il existe un algorithme **déterministe** en $\mathcal{O}(n^c)$ pour une constante c.

La classe de problèmes NP

- On peut donc définir une classe de complexité en fonction d'une **ressource** (*temps*, *espace*, etc.), une **borne** ($n^c, 2^{n^c}$, etc.) et un **mode** (*déterministe*, *non-déterministe*)
- On se restreint aux problèmes de **décision** : réponse dans $\{\text{oui}, \text{non}\}$

P

Ensemble des problèmes pour lesquels il existe un algorithme **déterministe** en $\mathcal{O}(n^c)$ pour une constante c.

NP

Ensemble des problèmes de **décision** pour lesquels il existe un algorithme **non-déterministe** en $\mathcal{O}(n^c)$ pour une constante c.

Problème de décision

- On se restreint aux **problèmes de décision**

Problème de décision Q

Fonction $Q : x \mapsto \{\mathbf{oui}, \mathbf{non}\}$

Problème de décision

- On se restreint aux **problèmes de décision**

Problème de décision Q

Fonction $Q : x \mapsto \{\mathbf{oui}, \mathbf{non}\}$

- Pour un problème dont la réponse n'est pas dans $\{\mathbf{oui}, \mathbf{non}\}$, on peut définir un problème **polynômialement** équivalent :

Problème de décision

- On se restreint aux **problèmes de décision**

Problème de décision Q

Fonction $Q : x \mapsto \{\mathbf{oui}, \mathbf{non}\}$

- Pour un problème dont la réponse n'est pas dans $\{\mathbf{oui}, \mathbf{non}\}$, on peut définir un problème **polynômialement** équivalent :

Voyageur de commerce (optimisation)

- **donnée** : ensemble de villes
- **question** : quel est le **plus court** chemin passant par toutes les villes ?

Problème de décision

- On se restreint aux **problèmes de décision**

Problème de décision Q

Fonction $Q : x \mapsto \{\mathbf{oui}, \mathbf{non}\}$

- Pour un problème dont la réponse n'est pas dans $\{\mathbf{oui}, \mathbf{non}\}$, on peut définir un problème **polynômialement** équivalent :

Voyageur de commerce (optimisation)

- **donnée** : ensemble de villes
- **question** : quel est le **plus court** chemin passant par toutes les villes ?

Voyageur de commerce (décision)

- **donnée** : ensemble de villes, entier k
- **question** : est-ce qu'il existe un chemin passant par toutes les villes de longueur inférieure à k ?

Problème d'optimisation, et sa version décision

Problème d'optimisation $Q = (X, f, m, g) : X \mapsto \bigcup_{x \in X} f(x)$

- X est un ensemble d'**instances**
- Pour tout instance $x \in X$, $f(X)$ est l'ensemble des solutions "faisables"
- $m : X \times f(X) \mapsto \mathbb{R}$ est une fonction "objectif"
- $g \in \{\min, \max\}$
- $Q(x) = arg_g\{m(x, y) \mid y \in f(x)\}$

Problème d'optimisation, et sa version décision

Problème d'optimisation $Q = (X, f, m, g) : X \mapsto \bigcup_{x \in X} f(x)$

- X est un ensemble d'**instances**
- Pour tout instance $x \in X$, $f(X)$ est l'ensemble des solutions "faisables"
- $m : X \times f(X) \mapsto \mathbb{R}$ est une fonction "objectif"
- $g \in \{\min, \max\}$
- $Q(x) = arg_g\{m(x, y) \mid y \in f(x)\}$

Exemple : Voyageur de commerce

- $x \in X$ est un ensemble de villes
- $y \in f(x)$ si et seulement si le chemin y passe par toutes les villes de x
- $m(x, y)$ est la longueur du chemin y (étant données les distances entre les villes de x)
- $g = \min$

Problème d'optimisation, et sa version décision

Problème d'optimisation $Q = (X, f, m, g) : X \mapsto \bigcup_{x \in X} f(x)$

- X est un ensemble d'**instances**
- Pour tout instance $x \in X$, $f(X)$ est l'ensemble des solutions "faisables"
- $m : X \times f(X) \mapsto \mathbb{R}$ est une fonction "objectif"
- $g \in \{\min, \max\}$
- $Q(x) = arg_g\{m(x, y) \mid y \in f(x)\}$

Problème de décision Q_D obtenu à partir de $Q = (X, f, m, \min)$

- **donnée** : $x \in X, k \in \mathbb{R}$
- **question** : Est-ce qu'il existe y tel que $f(x, y)$ et $m(x, y) \leq k$

Certificat polynômial

- On peut considérer des algorithmes non-déterministes de la forme suivante :

Certificat polynômial

- On peut considérer des algorithmes non-déterministes de la forme suivante :

Algorithme non-déterministe polynômial

- L'algorithme **devine** une solution y ou répond "non"
  - $\simeq$ pour toute solution possible, il existe une branche de l'arbre de calcul y menant

Certificat polynômial

- On peut considérer des algorithmes non-déterministes de la forme suivante :

Algorithme non-déterministe polynômial

- L'algorithme **devine** une solution y ou répond "non"
  - $\simeq$ pour toute solution possible, il existe une branche de l'arbre de calcul y menant
- La solution y doit être **vérifiable** en temps polynomial : *certificat*
  - $\simeq$ La longueur de chaque branche est polynômiale, i.e., en $\mathcal{O}(n^c)$

Certificat polynômial

- On peut considérer des algorithmes non-déterministes de la forme suivante :

Algorithme non-déterministe polynômial

- L'algorithme **devine** une solution y ou répond "non"
  - $\simeq$ pour toute solution possible, il existe une branche de l'arbre de calcul y menant
- La solution y doit être **vérifiable** en temps polynomial : *certificat*
  - $\simeq$ La longueur de chaque branche est polynômiale, i.e., en $\mathcal{O}(n^c)$

- Méthode simple pour déterminer si un problème appartient à la classe $\mathbf{NP}$: est-ce que toute réponse "oui" peut-être accompagnée d'un certificat (la solution) vérifiable en temps polynomial

Certificat polynômial : exemple

- Méthode simple pour déterminer si un problème appartient à la classe $\mathbf{NP}$: est-ce que toute réponse "oui" peut-être accompagnée d'un certificat (la solution) vérifiable en temps polynomial

Certificat polynômial : exemple

- Méthode simple pour déterminer si un problème appartient à la classe $\mathbf{NP}$: est-ce que toute réponse "oui" peut-être accompagnée d'un certificat (la solution) vérifiable en temps polynomial

Problème du voyageur de commerce

- **donnée** : ensemble de villes, entier k
- **question** : est-ce qu'il existe un chemin passant par toutes les villes de longueur inférieure à k ?

Certificat polynômial : exemple

- Méthode simple pour déterminer si un problème appartient à la classe $\mathbf{NP}$: est-ce que toute réponse "oui" peut-être accompagnée d'un certificat (la solution) vérifiable en temps polynomial

Problème du voyageur de commerce

- **donnée** : ensemble de villes, entier k
- **question** : est-ce qu'il existe un chemin passant par toutes les villes de longueur inférieure à k ?

- Certificat : le chemin (permutation de villes)
  - Vérifier que toutes les villes sont visitées est facile
  - Calculer la longueur totale est facile (somme)

Le problème complément

- Le **complément** d'un problème de décision : inversion des réponses "oui"/"non"

Le problème complément

- Le **complément** d'un problème de décision : inversion des réponses "oui"/"non"
  - Est-ce que le tableau T contient l'élement e ?
  - Est-ce que le tableau T **ne** contient **pas** l'élement e ?

Le problème complément

- Le **complément** d'un problème de décision : inversion des réponses "oui"/"non"
  - Est-ce que le tableau T contient l'élement e ?
  - Est-ce que le tableau T **ne** contient **pas** l'élement e ?
- Le complément d'un problème dans $\mathbf{P}$ est aussi dans $\mathbf{P}$
  - Le même algorithme peut être utilisé

Le problème complément

- Le **complément** d'un problème de décision : inversion des réponses "oui"/"non"
  - Est-ce que le tableau T contient l'élement e ?
  - Est-ce que le tableau T **ne** contient **pas** l'élement e ?
- Le complément d'un problème dans $\mathbf{P}$ est aussi dans $\mathbf{P}$
  - Le même algorithme peut être utilisé

Co-problème du voyageur de commerce

- **donnée** : ensemble de villes, entier k
- **question** : est-ce qu'il **n'**existe **aucun** chemin passant par toutes les villes de longueur inférieure à k ?

Le problème complément

- Le **complément** d'un problème de décision : inversion des réponses "oui"/"non"
  - Est-ce que le tableau T contient l'élement e ?
  - Est-ce que le tableau T **ne** contient **pas** l'élement e ?
- Le complément d'un problème dans $\mathbf{P}$ est aussi dans $\mathbf{P}$
  - Le même algorithme peut être utilisé

Co-problème du voyageur de commerce

- **donnée** : ensemble de villes, entier k
- **question** : est-ce qu'il **n'**existe **aucun** chemin passant par toutes les villes de longueur inférieure à k ?

- Ce problème est dans $\mathbf{NP}$, si et seulement si il existe un certificat polynômial, quel est le certificat polynômial ?

La classe coNP

Co-problème du voyageur de commerce

- **donnée** : ensemble de villes, entier k
- **question** : est-ce qu'il **n'**existe **aucun** chemin passant par toutes les villes de longueur inférieure à k ?

- Le co-problème du voyageur de commerce ne semble pas avoir de certificat polynômial

La classe coNP

Co-problème du voyageur de commerce

- **donnée** : ensemble de villes, entier k
- **question** : est-ce qu'il **n'**existe **aucun** chemin passant par toutes les villes de longueur inférieure à k ?

- Le co-problème du voyageur de commerce ne semble pas avoir de certificat polynômial
- Le complément d'un problème dans **NP** n'est souvent pas dans **NP** !

coNP

Ensemble des problèmes de décision dont le problème complément est dans **NP**

Inclusion dans EXPTIME

Inclusion dans EXPTIME

Considéront un problème dans **NP**

- La solution y doit avoir une taille polynômiale : $|y| \in \mathcal{O}(|x|^c)$

Inclusion dans EXPTIME

Considéront un problème dans **NP**

- La solution y doit avoir une taille polynômiale : $|y| \in \mathcal{O}(|x|^c)$
- L'arbre non-déterministe a donc au plus $\mathcal{O}(2^{|x|^c})$ feuilles

Inclusion dans EXPTIME

Considéront un problème dans **NP**

- La solution y doit avoir une taille polynômiale : $|y| \in \mathcal{O}(|x|^c)$
- L'arbre non-déterministe a donc au plus $\mathcal{O}(2^{|x|^c})$ feuilles
- Il est donc possible de les explorer séquentiellement en temps $\mathcal{O}(2^{|x|^c})$

Inclusion dans EXPTIME

Considéront un problème dans **NP**

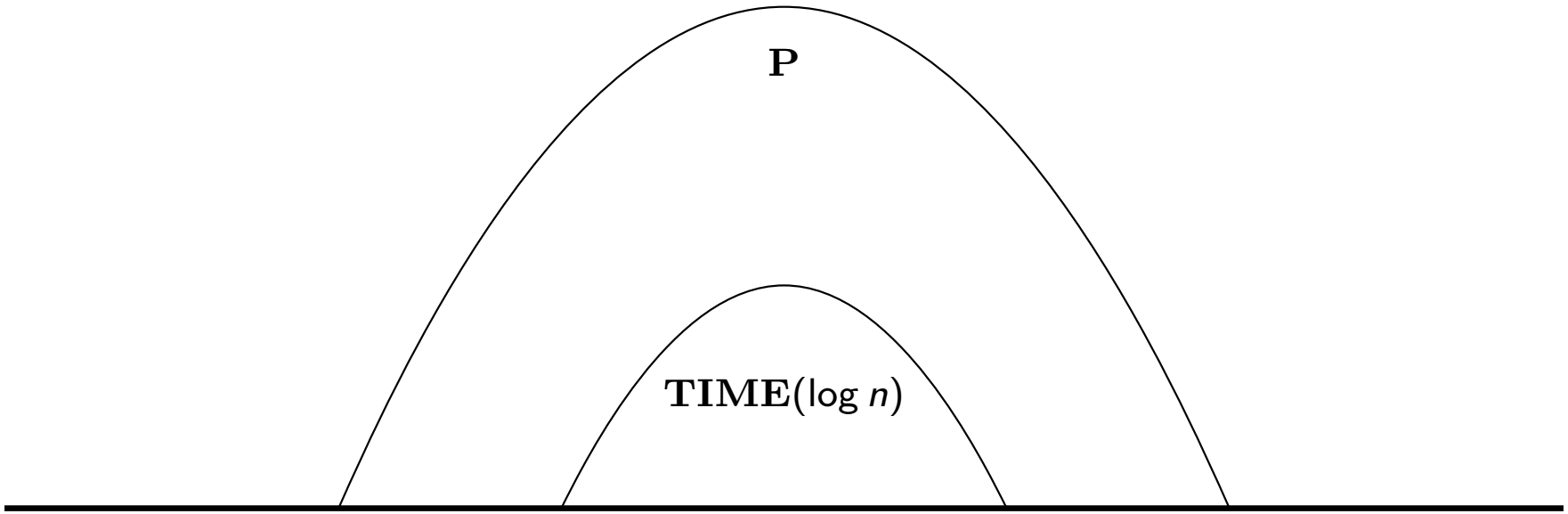
- La solution y doit avoir une taille polynômiale : $|y| \in \mathcal{O}(|x|^c)$
- L'arbre non-déterministe a donc au plus $\mathcal{O}(2^{|x|^c})$ feuilles
- Il est donc possible de les explorer séquentiellement en temps $\mathcal{O}(2^{|x|^c})$

Théorème

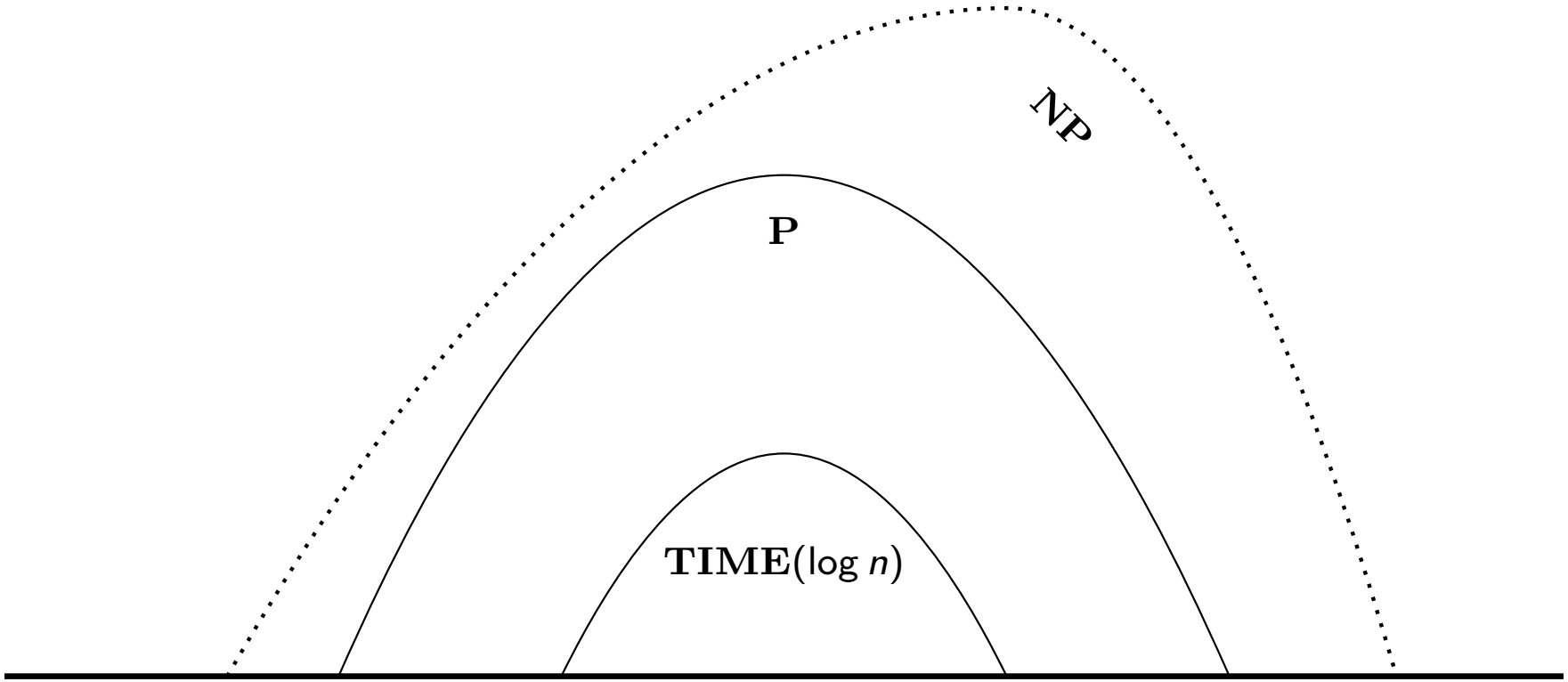
$$\mathbf{NP} \subseteq \mathbf{EXPTIME}$$

Résumé : les classes de complexité

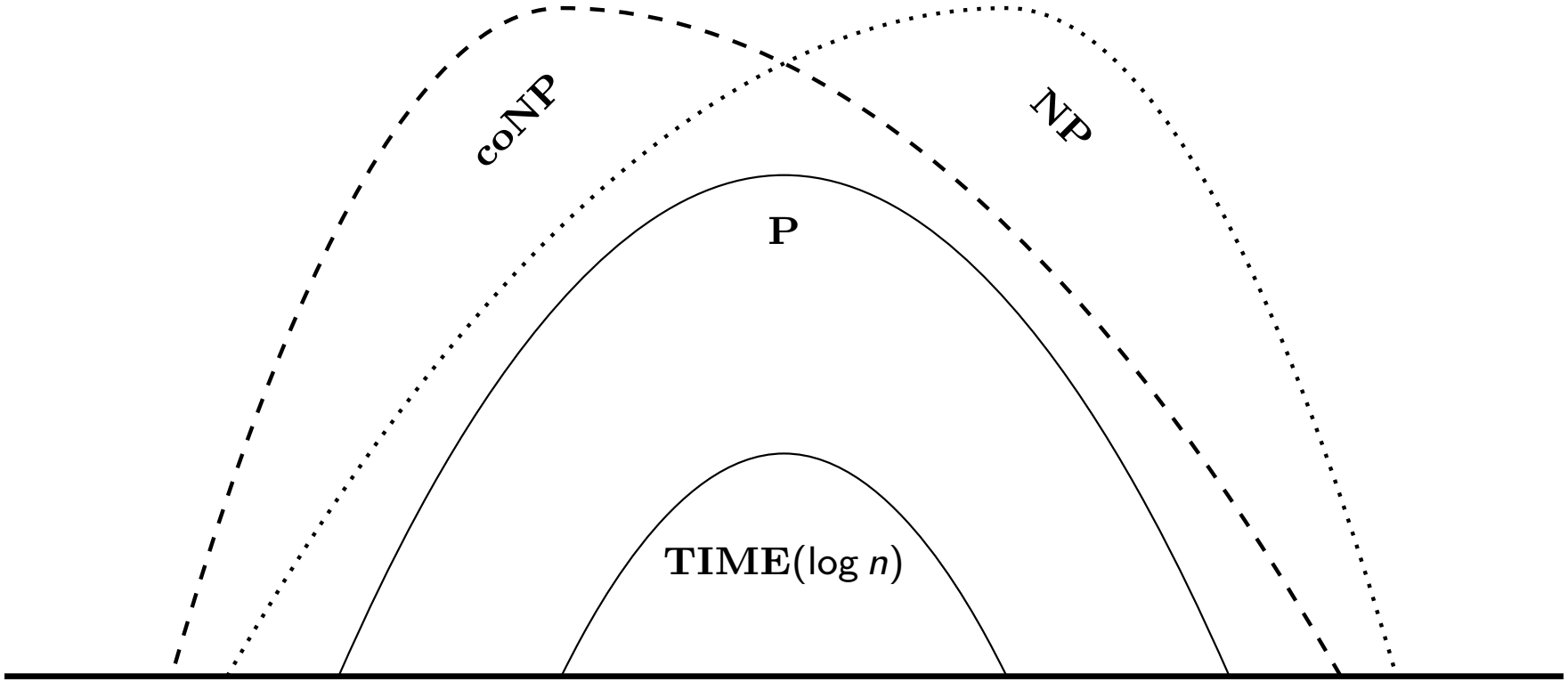
Résumé : les classes de complexité



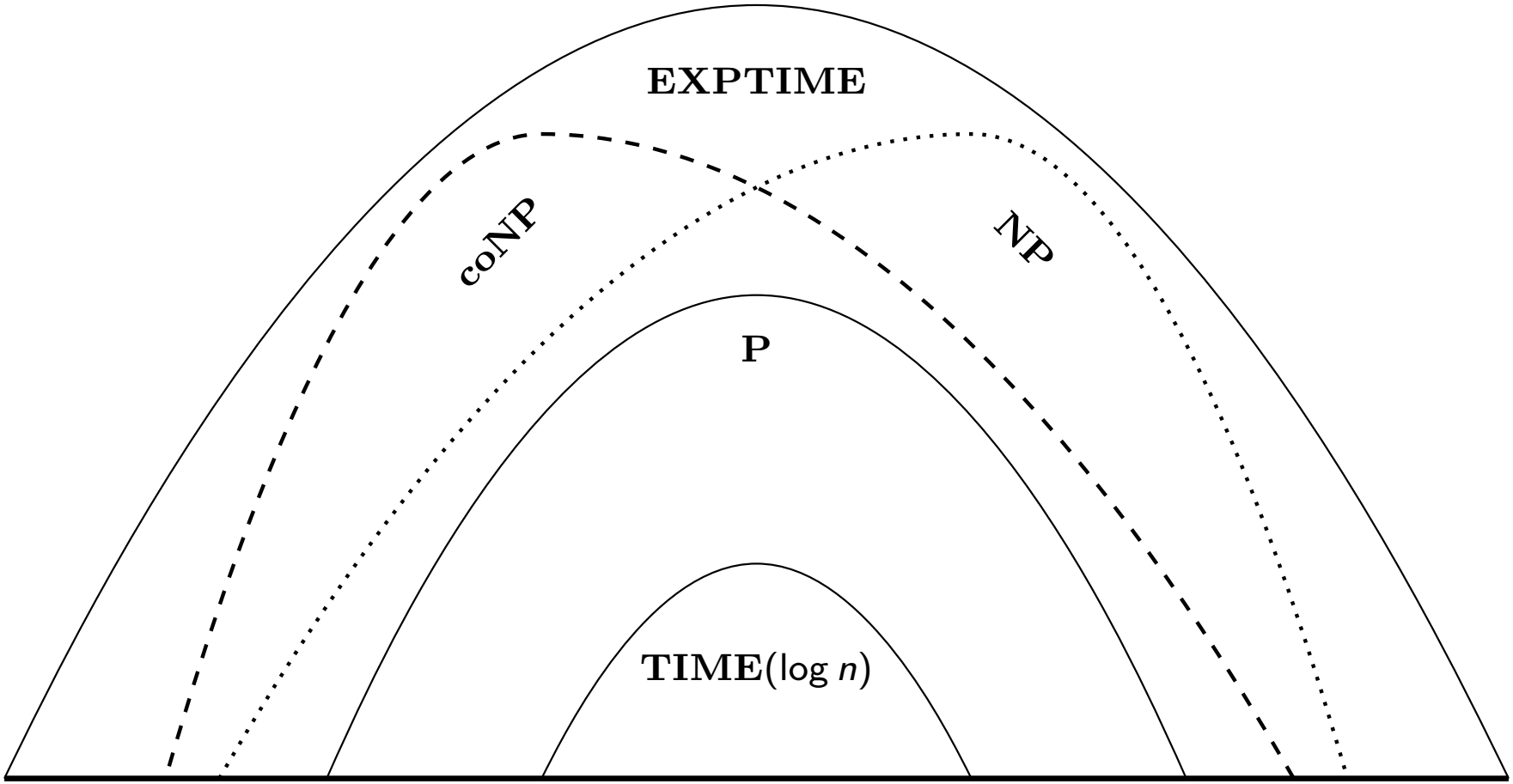
Résumé : les classes de complexité



Résumé : les classes de complexité



Résumé : les classes de complexité



L'importance de la classe NP

L'importance de la classe NP

- Ces problèmes sont partout : en intelligence artificielle, en cryptographie, dans l'industrie...

L'importance de la classe NP

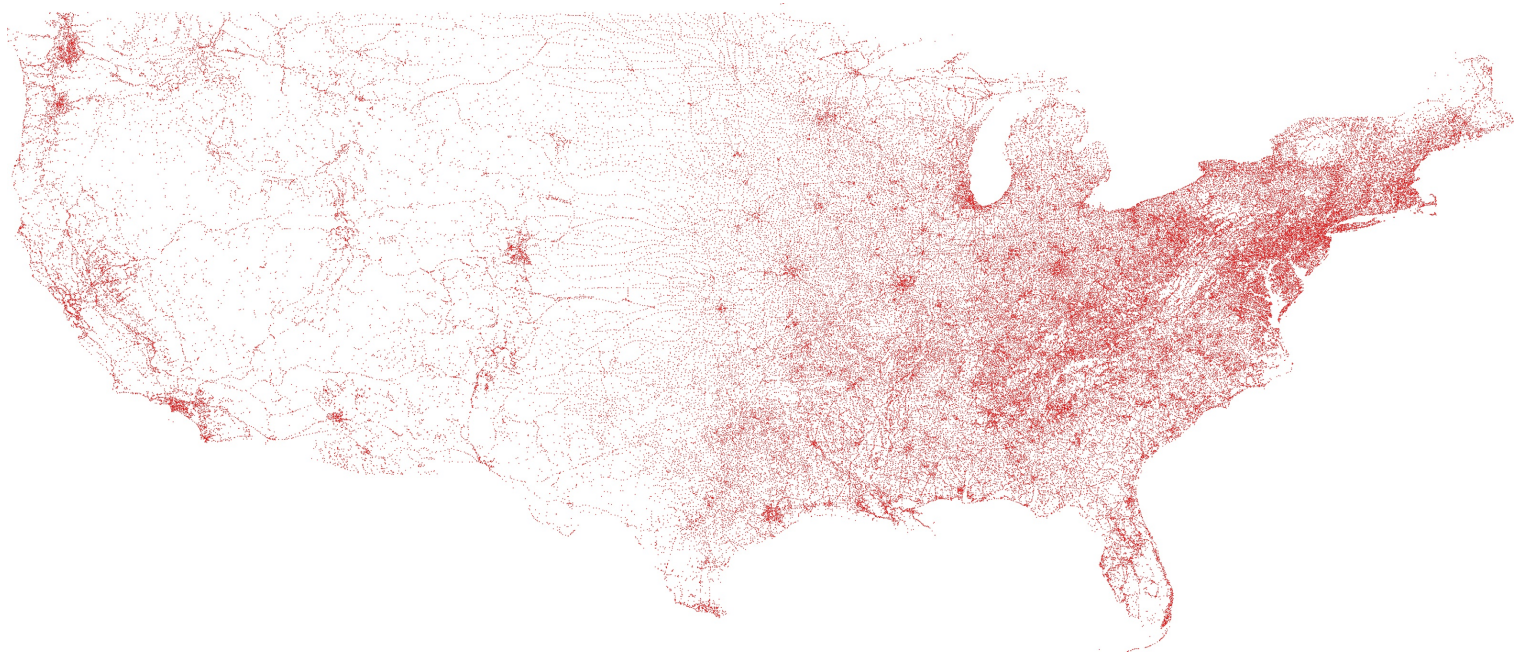
- Ces problèmes sont partout : en intelligence artificielle, en cryptographie, dans l'industrie...
- Savoir adapter la méthode à la complexité du problème ; le problème du voyageur de commerce n'a pas d'algorithme polynômial connu, mais...

L'importance de la classe NP

- Ces problèmes sont partout : en intelligence artificielle, en cryptographie, dans l'industrie...
- Savoir adapter la méthode à la complexité du problème ; le problème du voyageur de commerce n'a pas d'algorithme polynômial connu, mais...
  - Domaine de recherche très actif, des algorithmes "intelligent" peuvent résoudre (optimalement) de très grandes instances

L'importance de la classe NP

- Ces problèmes sont partout : en intelligence artificielle, en cryptographie, dans l'industrie...
- Savoir adapter la méthode à la complexité du problème ; le problème du voyageur de commerce n'a pas d'algorithme polynômial connu, mais...
  - Domaine de recherche très actif, des algorithmes "intelligent" peuvent résoudre (optimalement) de très grandes instances



115 475 villes

L'importance de la classe NP (suite)

L'importance de la classe NP (suite)

Conjecture $P \neq NP$

- Un des 7 "problèmes du millénaire" du Clay Mathematics Institute
  Mise-à-prix 1 000 000 $

L'importance de la classe NP (suite)

Conjecture $\mathbf{P} \neq \mathbf{NP}$

- Un des 7 "problèmes du millénaire" du Clay Mathematics Institute
  Mise-à-prix 1 000 000 $

- Preuve de $\mathbf{P} \neq \mathbf{NP}$: un problème dans $\mathbf{NP}$ mais pas dans $\mathbf{P}$

L'importance de la classe NP (suite)

Conjecture $\mathbf{P} \neq \mathbf{NP}$

- Un des 7 "problèmes du millénaire" du Clay Mathematics Institute
  Mise-à-prix 1 000 000 $

- Preuve de $\mathbf{P} \neq \mathbf{NP}$: un problème dans $\mathbf{NP}$ mais pas dans $\mathbf{P}$
  - Montrer que le problème est dans $\mathbf{NP}$ est facile : certificat polynômial
  - Montrer que le problème n'est pas dans $\mathbf{P}$ est difficile : tout algorithme est en $\Omega(2^n)$

L'importance de la classe NP (suite)

Conjecture $\mathbf{P} \neq \mathbf{NP}$

- Un des 7 "problèmes du millénaire" du Clay Mathematics Institute
  Mise-à-prix 1 000 000 $

- Preuve de $\mathbf{P} \neq \mathbf{NP}$: un problème dans $\mathbf{NP}$ mais pas dans $\mathbf{P}$
  - Montrer que le problème est dans $\mathbf{NP}$ est facile : certificat polynômial
  - Montrer que le problème n'est pas dans $\mathbf{P}$ est difficile : tout algorithme est en $\Omega(2^n)$

- Dans le prochain cours, on verra comment prouver que $\mathbf{P} = \mathbf{NP}$