Algorithmes certifiants

Michel Habib
habib@liafa.irif.fr
http://www.irif.fr/~habib

11 janvier 2018

Plan

Oublions les fêtes !

A good example

Kurt Mehlhorn and his group working on LEDA Library have a program for planarity testing :

- **YES** Answer = a planar drawing
- **NO** Answer = "This graph is not planar"

Only two years after they realized that the program has a flaw (or a bug)

A good progam for planarity testing :

- **YES** Answer = a planar drawing
- **NO** Answer = "an obstruction $K_{3,3}$ or K_5"

Based on Kuratowski's theorem, providing a certificate that can be checked in $O(n+m)$ in both cases.

How this idea can be expressed ?

Main ideas = certificate and testing.
One can find in the literature two close notions

First a pragmatic way

Algorithms with certificates easy to check. Certifying algorithms
Easy = polynomial.

The second approach coming from algorithmic complexity

Les (Existencially Provable) EP theorems, J. Edmonds 1990.
Robusts algorithms, J. Spinrad 2002.

Main references

- K. Cameron, J. Edmonds, Existencially polytime theorems, Dimacs Series in DMTCS, (1990), 83-100.
- D. Kratsch, R.M. Connell, K. Mehlhorn, J. Spinrad, Certifying algorithms for recognizing interval graphs and permutation graphs, SODA 2003.
- V. Raghavan, J. Spinrad, Robust algorithms for restricted domains, J. of Algorithms 48 (2003) 160-172.
- J. Spinrad, Efficient graph representations, Fields Institute Monographs, 2003.
- H. Wasserman, M. Blum, Software reliability via run-time Result checking, JACM 44 (1997) 826-849.

Certificate versus Proof

Each time a program is used, one can check its result by testing a certificate given by the algorithm

Else we should :

- Prove the algorithm (using invariants)
- Proving the transformation from an algorithm to a program
- Prove the program itself (Data structures . . .)
- Be confident to (or prove) the compiler, the Operating System . . .

Notion de certificat

Rappelons qu'un certificat est une chaîne de caractères qui permet de vérifier le résultat d'un calcul. Par exemple si un programme répond qu'un nombre entier n est composé, deux entiers p, q tels que $n = pq$ constituent un certificat du fait que n ne soit pas premier.

Il s'agit de proposer des certificats pour tous les problèmes algorithmiques. Dans chacun des cas, on précisera la taille du certificat en fonction de celle de la donnée, ainsi que la complexité de l'algorithme permettant de calculer le certificat ainsi que celle de l'algorithme qui permet de vérifier le certificat.

C'est la même notion que intervient en théorie de la complexité avec l'idée de certificat polynomial.

Le premier des algorithmes

$Pgcd(a, b)$
Comment certifier le résultat ?

Et pour le tri ?

Il est facile de vérifier la monotonie du résultat.
Mais les ensembles données résultats sont-ils bien les mêmes ?

Recherche de l'élément médian

Entrée : un tableau d'entiers positifs A

- Résultat : x médian
- Résultats : x et B, C une partition de A-$\{x\}$ en deux sous tableaux vérifiant :
  $x \in A$, tel que : $B < x$ et $x \leq C$
- Certificat possible dans le deuxième cas en $|A|$ ou $|B|$. Dans le premier cas si l'on veut vérifier la réponse, il faut comparer x à tous les éléments de A.

- Vérifier une multiplication d'entiers de 64 bits
- Composantes connexes, composantes fortement connexes

2-coloration

YES answer : a bipartition of the vertices into 2 independent sets $O(n+m)$

NO answer : an odd cycle $O(n)$

Bad cases

When the certificate is the algorithm itself.

Good cases

The two certificates for YES and NO Answers can be checked independently from the algorithm.

Very good cases

Algorithms **very easy to check** : the certificates can be tested within an algorithmic complexity not greater than the algorithm

Some examples

- 2-colorable graphs (bipartite) $(O(n+m), O(n))$.
- Cographs ou P_4-free graphs $(O(n+m), O(1))$.
- Interval graphs , permutation graphs $(O(n+m), O(n))$. (Kratsch et al 2003)

Minimum spanning trees

A spanning tree can be produced in $O(n + mlogn)$

Checking its minimality can be done in $O(n + m)$ (Good exercise)

Shortest paths SSSP

Dijkstra's algorithm computes in $O(n + mlogn)$ a tree T rooted in s providing a path form s to the others vertices.

The minimality of T can be checked in $O(n + m)$.
$\forall e = (x, y) \in G - T$, $d_T(y) \geq d_T(x) + \omega(e)$

Characterisitic linear ordering of the vertices

Many graph algorithms can be seen as the computation of some characteristic ordering of the vertices.
Examples : simplicial elimination scheme, transitive orientation, chordal graphs, interval graphs, unit interval gaphs, permutation graphs, cographs, distance-hereditary graphs, factoring permutation for modular decomposition. ...

2-step algorithms

1. Computation of an ordering of the vertices suposed to have some property α
2. Testing the property α.

These algorithms often produces certifying algorithms

Extensions

- Similar questions for automaton (par ex : minimal automaton)
- Same notion for enumerating algorithms.
- Probabilistic Certificates (N. Alon)
- Related works for hardware : Software reliability via run-time result checking, H. Wasserman, M. Blum, JACM 1997

Another way to consider algorithms

Let us apply these ideas to well-know problems (example searching in an ordered array).

The game is to obtain the lowest complexity

R McConnell, K. Mehlhorn, S. Näher and P. Schweitzer are writing a book on this subject :
htpp ://www.mpi-inf.mpg.de/ mehlhorn/ftp/CertifyingAlgorithms.pdf

Each time you write an algorithm, ask yourself :
Does there exists another way to validate the result ?

From algorithmic complexity theory

Only for decision problems

The symmetry of the answers YES–NO force us to consider only $NP \cap co - NP$

It is hard to certify that the value given by some heuristic is less than K-times the optimum value.

To compute a graph parameter k, as for example treewidth, we need an algorithm which produces either a value $\leq f(k)$, or a certificate than the certificate is greater than k (using Brambles for treewidth).

Good characterizations

NP is the class of decision problems with a polynomial certificate for the YES Instances.

$NP \cap co - NP$ polynomial certificate in both cases
(Already in the first Jack's ideas in 1965)

Let us only consider only graph problems to illustrate (cf. using Fagin's characterization theorems for P et NP).

Famous conjecture

$P = NP \cap co - NP$? had important consequences.

- Linear Programming (Kachian, 1979)
- Primality testing (2002)
- Perfect Graphs recognition (2003)
- Parity Games ?
- Minimal Transversal ?

EP theorems, J. Edmonds 1990

An EP (Existentially Polytime) theorem is a theorem in which each condition is polynomially testable.

Exemples :

- A good characterisation
- un graph is not perfect iff it contains an odd hole or its complement.

EP theorems

Min-Max theorems

Flow max = min cut

An optimal cut provides a certificate to a flow

Another example

(Either they are k edge-disjoint paths from a to b in G)
(or their is a cut of size k-1 separating a from b in G)
(But not both).

Marriage problem

For any input of the problem :
either there is a way for all the girls to marry distinct boys who love them,
or else there is a subset S of the girls which is bigger than the number of boys who love someone in S.
Froebenius had already found this theorem.

The best way to prove an EP theorem is to give an algorithm.
Example : Hungarian method for the optimum matching problem

C. Berge and Jack Edmonds

EP theorems

Sans vraiment l'écrire explicitement, J. Edmonds pense qu'un tel théorème implique l'existence d'un algorithme polynomial (au moins ceux du type $NP \cap co - NP$).
Exemple : La reconnaissance des graphes parfaits (2003).

Précisions sur les questions du dernier cours

NP versus EXP

NP la classe des problèmes de décision ayant un algorithme polynomial sur une machine de Turing non déterministe
$\subsetneq$
EXP la classe des problèmes de décision ayant un algorithme exponentiel sur une machine de Turing déterministe

Conjectures

Même si elles sont reliées, les conjectures suivantes ne sont pas équivalentes :

- $NP = co\text{-}NP$?
- $P \neq NP$?
- $NP \cap co\text{-}NP = P$?

Faits

Il existe un algorithme en $O(n^{logn})$ pour l'isomorphisme de deux graphes.
Personne ne connaît un tel algorithme pour un problème NP-complet.

Hiérarchie de problèmes

Une hiérarchie infinie en P et NP ?

n^{logn}, n^{logn^2} ... n^{logn^i} ...

Comments

Transversal minimal $\in P$?

Data : G a bipartite graph, $\mathcal{T}$ a set of transversal of G

Question : Is $\mathcal{T}$ the set of all minimal transversals of G ?

- A transversal is simply a set of vertices intersecting all edges.
- Transversal minimal $\in NP \cap co\text{-}NP$
- Kachyan proposed an algorithm in $O(n^{logn})$.
- A hot subject ! ! !

Intuition *P* versus *NP*-complet

- Lorsque le problème est NP-complet, l'espace des solutions (prenons l'exemple de SAT) n'a pas de structure connue exploitable par un algorithme.
- Lorsque le problème est polynomial, il existe une certaine structure algébrique de l'ensemble des solutions (arbres de poids min : matroïdes, couplages : intersection de deux matroïdes, programmation linéaire (polyèdres et matroïdes orientés), plus courts chemins : algèbres de chemins, algèbres tropicales ...).

Pour chaque algorithme polynomial important du cours, nous avons rencontré une structure algébrique :

- arbres – matroïdes
- couplages – intersection de deux matroïdes
- flots – lemme de Minty, algèbre linéaire avec Kirchoff
- Chemins – algèbre des chemins
- 2-SAT – chemins dans un graphe fortement connexe (cela ne marche plus à 3-SAT)