2nd Lecture : Basic techniques for graph algorithms MPRI 2013–2014

Michel Habib
habib@liafa.univ-Paris-Diderot.fr
http://www.liafa.univ-Paris-Diderot.fr/~habib

Sophie Germain, septembre 2013

Schedule

Introduction

Good algorithms and graph decompositions

Graph Representations

Cograph recognition : the naive approaches

Partition refinement techniques

Co-graph recognition problem as a leading example (in french "fil rouge") to present various algorithmic techniques.
Nota Bene : the slides will be the unique official document of the course. If you solve all the exercises, your will be safe for the exam.

An example of nice algorithm

Depth First Search

DFS(G) :
- **for all** $v \in X$ **do**
  - $Ferme(x) \leftarrow Faux$;
- **end for**
- **for all** $v \in X$ **do**
  - **if** $Ferme(x) = Faux$ **then**
    - $Explorer(G, x)$;
  - **end if**
- **end for**

Explorer(G, x) :
- $Ferme(x) \leftarrow Vrai$;
- $pre(x)$;
- **for all** $xy \in U$ **do**
  - **if** $Ferme(y) = Faux$ **then**
    - $Explorer(G, y)$;
  - **end if**
  - $post(x)$;
- **end for**

Et les deux fonctions suivantes utilisant deux variables : comptpre et comptpost étant initialisées à 1.

pre(x) :
- $pre(x) \leftarrow comptpre$;
- $comptpre \leftarrow comptpre + 1$;

post(x) :
- $post(x) \leftarrow comptpost$;
- $comptpost \leftarrow comptpost + 1$;

Tarjan's algorithm to compute strongly connected components

DFS(G) :
- $comptpre \leftarrow 1$;
- $Resultat \leftarrow \emptyset$;
- **for all** $v \in X$ **do**
  - $Ferme(x) \leftarrow Faux$;
- **end for**
- **for all** $v \in X$ **do**
  - **if** $Ferme(x) = Faux$ **then**
    - $Explorer(G, x)$;
  - **end if**
- **end for**

Explorer(G, x) :
- $Empiler(Resultat, x)$;
- $Pile(x) \leftarrow Vrai$; $Ferme(x) \leftarrow Vrai$;
- $pre(x) \leftarrow comptpre$; $comptpre \leftarrow comptpre + 1$;
- $racine(x) \leftarrow pre(x)$;
- **for all** $xy \in U$ **do**
  - **if** $Ferme(y) = Faux$ **then**
    - $Explorer(G, y)$;
    - $racine(x) \leftarrow min\{racine(x), racine(y)\}$;
  - **else if** $Pile(y) = Vrai$ **then**
    - $racine(x) \leftarrow min\{racine(x), racine(y)\}$;
  - **end if**
- **end for**
- **if** $racine(x) = pre(x)$ **then**
  - Dépiler Resultat jusqu'à x ;
- **end if**

Fundamental questions about algorithms

1. How can we prove such a nice algorithm ?
2. It has some greedy flavor, why ?

Our Claims or thesis

An efficient algorithm running on a discrete structure is always based :

- on a theorem describing a combinatorial structure
- a combinatorial decomposition of this discrete structure.
- or in some other cases a geometric representation of the structure provides the algorithm.

Examples

- Chordal graph recognition and maximal clique trees (particular case of **treewidth**).
- Transitive orientation and modular decompostion.
- Max Flow and decomposing a flow in a sum of positive circuits.
- Greedy algorithms for minimum spanning trees and matroids

Interval graphs

Let us consider an operation research problem :

- Storage of products in fridges : each product is given with an interval of admissible temperatures.
  Find the minimum number of fridges needed to store all the products (a fridge is at a given temperature).
- A solution is given by computing a minimum partition into maximal cliques.
- Fortunately for an interval graph, this can be polynomially computed
- So knowing that a graph is an interval graph can help to solve a problem.

Interval graphs are used to modelize time (in scheduling) but also to analyze DNA sequences.

Notations

Here we deal with finite loopless and simple undirected graphs.
For such a graph G
we denote by $V(G)$ the set of its vertices
and by $E(G)$ the edge set
By convention $|V| = n$ and $|E| = m$

Bound on the number of edges

Triangle free graphs

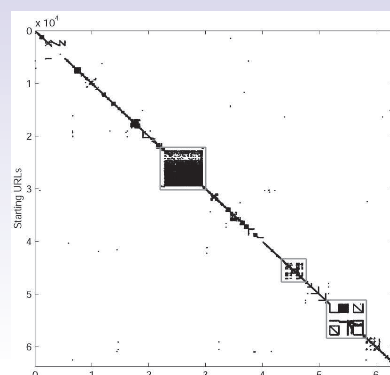
Show that if G has no triangle then :
$|E| \leq \frac{|V|^2}{4}$

Planar graphs

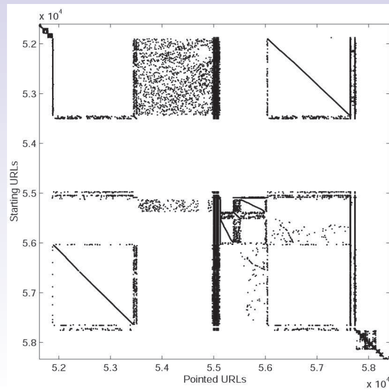
Show that if G is a simple planar graph (i.e. without loop and parallel edge) then :
$|E| \leq 3|V| - 6$

Sparse graphs satisfy : $|E(G)| \in O(|V(G)|)$
Planar graphs are sparse, but also many graphs coming from applications are sparse.
For example the WEB graph is sparse.
For sparse graphs one has to use an adjacency lists representation.

Matrice ordonnée par l'ordre alphabétique des noms des URL



Un zoom autour de la diagonale



- Implicit hypothesis : the memory words have k bits with $k > \lceil log(|V|) \rceil$
- To be sure, consider the bit encoding level

- Adjacency lists
  $O(|V| + |E|)$ memory words
  Adjacency test : xy is an arc in $O(|N(x)|)$
- Adjacency Matrix
  $O(|V|^2)$ memory words (can be compressed)
  Adjacency test : xy is an arc in $O(1)$
- Customized representations, a pointer for each arc ...

For some large graphs, the Adjacency matrix, is not easy to obtain and manipulate.
But the neighbourhood of a given vertex can be obtained. (WEB Graph or graphs is Game Theory)

Quicksands

- A sentence like :
  "To compute this invariant or this property of a given graph G one needs to "see"(or visit) every edge at least once".
- False statement as for example the computation of twins resp. connected components on $\overline{G}$ knowing G.

Exercise

Can the advantages of the 2 previous representations can be mixed in a unique new one ?
Adjacency lists : construction in $O(n + m)$
Incidence matrix : cost of the query : $xy \in E$? in $O(1)$

In other words
Using $O(n^2)$ space, but with linear time algorithms on graphs ?

Auto-complemented representations

Initial Matrix

	1	2	3	4
1	1	1	1	0
2	0	0	1	0
3	1	0	1	1
4	1	0	0	0

Tagged Matrix

	$\overline{1}$	2	$\overline{3}$	4
1	0	1	0	0
2	1	0	0	0
3	0	0	0	1
4	0	0	1	0

- At most $2n$ tags (bits).
  $O(n + m')$ with $m' << m$.
  Dalhaus, Gustedt, McConnell 2000
- What can be computed using such representations ?
  Example : strong connected components of G, knowing $\overline{G}$?

- Find a minimum sized representation
- If G is undirected a minimum is unique.

What is an elementary operation for a graph ?

- Traversing an edge or Visiting the neighbourhood ?
- It explains the very few lower bounds known for graph algorithms on a RAM Machine.
- Our graph algorithms must accept any auto-complemented representation.
  Partition Refinement

Exercise

Suppose that a graph $G = (V, E)$ is given by its adjacency lists and let σ be some total ordering of its vertices.

- How can we sort the adjacency with respect to σ (increasing) ?
- What is the complexity of this operation ?
- Let σ be the total ordering of the vertices with decreasing degrees, how to compute σ ?

Sorting an adjacency list

Suppose that a graph $G = (V, E)$ is given by its adjacency lists A and let σ be some total ordering of its vertices. How can we sort the adjacency with respect to σ (increasing) ?

One solution

Build another adjacency list structure B from the old one by taking the vertices from $\sigma(n)$ down to $\sigma(1)$ in the following way : read $A(\sigma(i))$ and add $\sigma(i)$ in front of the lists of B corresponding to the neighbors of $\sigma(i)$
At the end of the process, the lists of B are sorted in the right way.

Complexity

Time : Linear time complexity (the size of the data structure A).
Memory : Twice the size of the adjacency lists $2|A|$

Exercise

Adapt the solution for directed graphs

Sorting the vertices by their degrees

1. Compute the degrees by scanning the adjacency lists in an array D
2. Use any per value sorting algorithm (Radix or other) since all numbers are bounded by n for a simple graph.
3. Apply the previous algorithm to sort the adjacency lists.

The previous solutions need that the data structure is available at once in the memory.
This will be implicit in the remaining of the course, as well as the RAM model.

How to find an induced P_4 ?

- Very naive $O(n^4)$:
  Check every quadruple of vertices
- Less naive $O(mn^2)$:
  For every edge $xy \in E(G)$
  For every pair (z, t) of vertices with $z \in N(x) - N(y)$ and $t \in N(y) - N(x)$
  Check if $zt \notin E(G)$.

- $O(m^2)$ [1]
  For every pair of edges : $zx, ty \in E(G)$
  Check in $O(1)$ if they are the external edges of some P_4
  (using the incidence matrix representation)
- $O(nm)$
  For every $z \in V(G)$ compute $N(z)$ and $\overline{N(z)}$
  Check if there exist $x \in N(z)$ and one edge $zt \in \overline{N(z)}$
  such that x splits (or distinguishes) z and t, using partition refinement techniques.

1. Could be interesting for sparse graphs.

Exercises

1. Show that if G is a cograph, then either G is connected or disconnected and the 2 cases are exclusive.
2. How to compute Max Cut on a cograph ?
3. How to recognize if a graph is P_5-free ?

First linear-time algorithm

Derek G. Corneil and Yehoshua Perl and Lorna K. Stewart
A Linear Recognition Algorithm for Cographs,
SIAM J. Comput.,14, 4 (1985) 926-934.

Sketch of the algorithm

1. Uses the existence of the cotree and incrementally build the cotree.
2. At each step a new vertex x_i is added.
3. Compute the minimal subtree A_i whose leaves contains $N(x_i) \cap G(\{x_1, \dots, x_{i-1}\})$.
4. If A_i has more leave than $N(x)$ STOP "G is not a cograph" [2].
   Else if the root of A_i is labeled 1 add x_i as a leave just attached to this node.
   Else consider the parent node of A_i (labeled 1), it has at least another child B. Add x_i as a leave just attached to the root of B.

2. this is a simplification, for details see the original paper

- STEP 3 can be done in $O(|N(x_i)|)$, so a total complexity of $O(n+m)$.
- When it fails, x_i belongs to a P_4. (Case by case analysis).
- Comments : the existence of the cotree is very useful !

Definition

Partition Refinement

If $Q = \{C_1, \dots C_k\}$ is a partition over a ground set X, for every $S \subseteq X$ we define from Q and S a new partition :
$Refine(Q, S) = \{C_1 \cap S, C_1 - S, \dots, C_k \cap S, C_k - S\}$ [a]

a. empty sets are removed

More formally

$Refine(Q, S) = Q \wedge \{S, X\text{-}S\}$ in the partition lattice on X.

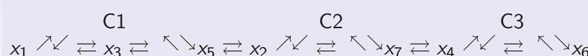
The method

The partition P is made up with a list of classes.
For each element x of the pivot set S, find the unique part C it belongs to.
Then move (or mark) x in C and tag C
At last, separate all marked parts C into $C \cap S$ and C-S.

Implementation

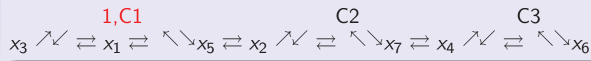
$V = \{x_1, \dots, x_7\}$
$P = \{C1, C2, C3\}$ and $S = \{x_3, x_4, x_5\}$

Data Structure

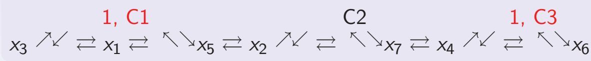


First Step

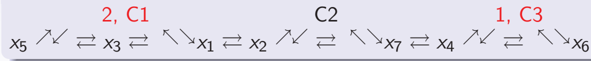
Processing x_3



Processing x_4



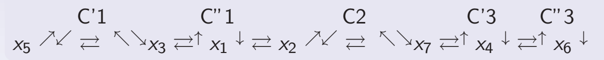
x_5



Second step

Maintain a list of the $Ci's$ that intersect S. List bounded by $|S|$.

Result



$Refine(P, S) = \{C'1, C"1, C2, C'3, C"3\}$
computed in $O(|S|)$

To implement this data structure we need at least :

- A doubled linked list L for the partition itself
- For each element of V, we need to maintain a reference to its position in the list
- For every element in L we need to maintain a reference to its part.
- Every part has to maintain a reference to its first element.

Of course it could be implemented using arrays instead of linked data structures. Furthermore this could be much more efficient in many programming languages (for example those in which list are badly implemented).
This technique is not only efficient theoretically (with respect to complexity measures) but also for practical purpose, since this technique can be implemented with a small overhead.

Exercises

1. Propose an alternative implementation using arrays.
2. Propose an implementation in $O(|S|)$ of Refine stable which preserves a given initial ordering of the vertices of the elements $x_i's$ of the parts.
3. Propose an implementation in an array compatible with an initial ordering and within the same complexity.