Chordal graphs: a linear testing algorithm

Gabriele Vanoni

Politecnico di Milano

23 May 2017

The algorithm

Preliminary definitions

Definition (Undirected graph)

A *graph* G is a pair (V, E) where V is a finite set and E is a set of $2-$subsets of V. Elements of V are called *vertices* while elements of E are called *edges*. $|V| = n$, $|E| = m$.

Definitions (Path, Cycle, Chord)

- A *path* π is a sequence of distinct vertices $v_1, v_2, \cdots v_i, \cdots, v_k$ where $\{v_i, v_{i+1}\} \in E$ with $1 \leq i < k$.
- A *cycle* of lenght $k+1$ is a closed path, i.e. a path for which $\{v_1, v_k\} \in E$.
- A *chord* is an edge connecting two nonconsecutive vertices of a cycle.

Definition (Chordal graph)

A graph is chordal if every cycle of lenght at least four has a chord.

Orderings and fill-ins

Definition (Ordering)

An *ordering* is a bijection $\alpha : V \rightarrow \{1, 2, \cdots, n\}$. $v <_\alpha w$ iff $V(v) < V(w)$.

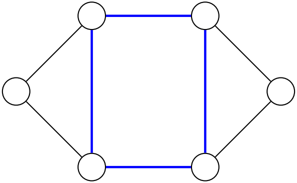
Definition (Fill-in)

A *fill-in* induced by an ordering α is a set of edges $F(\alpha) \notin E$ such that there exists a path containing only u, v and vertices ordered after both u and v. $F(\alpha)$ is *zero fill-in* if $F(\alpha) = \varnothing$ and α is a zero fill-in ordering.

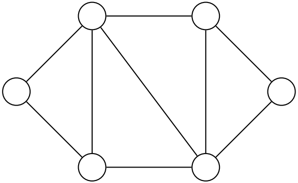
Definition (Elimination graph)

The *elimination graph* of G w.r.t. the ordering α is $G(\alpha) = (V, E \cup F(\alpha))$.

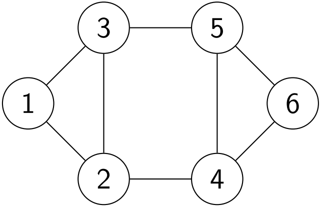
Example



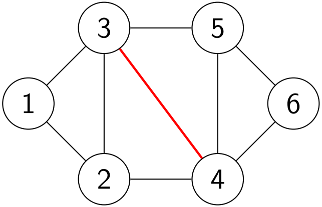
not chordal



chordal



ordered



elimination graph

Testing chordality

Theorem

A graph is chordal if and only if it has a zero fill-in ordering.

We want to compute an ordering α that is zero fill-in if and only if G is chordal. In this way we can compute $F(\alpha)$. G is chordal if and only if $F(\alpha) = \varnothing$.

Definition (Maximum cardinality search (MCS))

It's an ordering algorithm in which at each step i (from 1 to n) the vertex selected and numberd with i among the unnumbered ones is that adjacent to the largest number of previously numberd vertices, breaking ties arbitrarily.

Theorem

An ordering generated by MCS is zero fill-in if the graph is chordal.

Testing chordality - complexity analysis

Theorem

Complexity of the algorithm that computes the MCS ordering is $\mathcal{O}(n+m)$.

Proof.

The first two for loops are $\mathcal{O}(n)$. The third is also $\mathcal{O}(n)$ but there are inner loops. However in the first one every edge is scanned at most twice, while the second one is executed at most n times yielding a total cost of the outer loop of $\mathcal{O}(n+m)$. □

for $i = 0$ **to** $n-1$ **do**
  $set[i] := \varnothing$
end for
for all v in V **do**
  $size[v] := 0$; $set[0] := set[0] \cup \{v\}$
end for
$j := 0$
for $i = 1$ **to** n **do**
  $v :=$ delete any node from $set[j]$
  $\alpha[v] := i$; $\alpha^{-1}[i] := v$; $size[v] := -1$
  for all $(u, v) \in E$ and $size[u] \geq 0$ **do**
    delete u from $set[size[u]]$
    $size[u] := size[u] + 1$
    $set[size[u]] := set[size[u]] \cup \{u\}$
  end for
  $j := j + 1$
  while $j \geq 0$ and $set[j] = \varnothing$ **do**
    $j := j - 1$
  end while
end for

Computing the fill-in

Definition

The *follower* of a vertex v, $f(v)$ is the vertex w of largest number (w.r.t to α) both adjacent to v in $G(\alpha)$ and such that $w <_\alpha v$. For $i \geq 0$, $f^0(v) = v$ and $f^{i+1}(v) = f(f^i(v))$.

Theorem

If $x, w \in V$ with $w <_\alpha x$, then $(x, w) \in E \cup F(\alpha)$ if and only if there is a vertex v such that $(v, w) \in E$ and $f^i(v) = x$ for some $i \geq 0$.

With this theorem we can compute for any vertex w, the set $A(w) = \{x | (x, w) \in E \cup F(\alpha), w <_\alpha x\}$ and also all vertices x such that $f(x) = w$.

Computing the fill-in - complexity analysis

Theorem

Complexity of the algorithm that computes the fill-in of a graph G is $\mathcal{O}(n + m')$ where $m' = |E \cup F(\alpha)|$.

Proof.

The outer loop is executed n times. The inner one is scans each vertex of the elimination graph at most twice yielding a total cost of the outer loop of $\mathcal{O}(n + m')$. □

for $i = n$ **to** 1 **do**
    $w := \alpha^{-1}[i]$
    $f[w] := w$
    $index[w] := i$
    for all $v \in V$ s.t. $(v, w) \in E$ and $\alpha[v] > i$
        $x := v$
        while $index[x] > i$ **do**
            $index[x] := i$
            add (x, w) to $E \cup F(\alpha)$
            $x := f[x]$
        end while
        if $f[x] = x$ **then**
            $f[x] := w$
        end if
    end for
end for

Implementation

Language and libraries

- JavaSE 8 was used to implement the algorithm.
- Graph data structures were provided by JUNG library and graphs were stored in PajekNet format.
- Apache Maven was used to handle the project.
- Eclipse was used as IDE.
- The code is released under GPL 3 license.
- Oracle Java Mission Control with Flight Recorder was used to profile the application.

Data structures

Two data structures were defined to implement the algorithms.

Vertex

- `alpha:int`
- `size:int`
- `index:int`
- `follower:Vertex`

ChordalAlgorithms

- `order:List<Vertex>`
- `graph:UndirectedGraph<Vertex,Integer>`
- `maximumCardinalitySearch():void`
- `isChordal():boolean`

Testing environment

- **JVM**: Oracle 1.8.0_111-b14
- **OS**: Ubuntu 15.10
- **Linux kernel**: 4.2.0-42-generic
- **libc**: glibc 2.21
- **System architecture**: x86_64
- **CPU**: Intel Core i7-2600K CPU @ 3.40GHz
- **Memory**: 8 GB

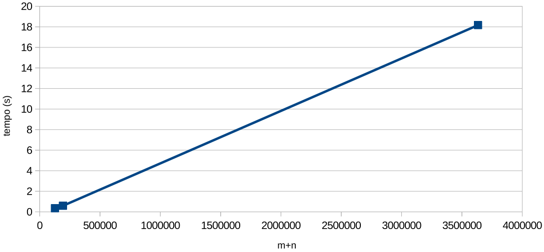
Datasets

- I chose to use real networks data to test the algorithms.
- All graphs from datasets were obviously non chordal.
- For every datasets, I computed the elimination graph, so to have a chordal graph and I exported it.
- To measure performances I used only chordal graphs so that the complexity is $\mathcal{O}(n+m)$.

Profiling

Total time is the actual time spent by the application to execute the two methods of the algorithm, `maximumCardinalitySearch()` and `isChordal()`. The data structure `ChordalAlgorithms` was already filled in with the graph read from file.

Dataset	n+m	Total time
YeastChordal	126547	345 ms
PGPChordal	192058	586 ms
DaysAllChordal	3633865	18s 166 ms



Parallel versions

Parallel versions of the algorithm

Recognizing is in the complexity class **NC**, as shown for the first time by Edenbrandt (1986), using a different characterization of chordal graphs. After this seminal paper other results were achieved in optimizing the algorithm.

Authors	Complexity	Number of processors	Cost model
Edenbrandt	$\mathcal{O}(\log n)$	$\mathcal{O}(n^5)$	PRAM CREW
Chandrasekharan et al.	$\mathcal{O}(\log n)$	$\mathcal{O}(n^4)$	PRAM CRCW
Klein	$\mathcal{O}(\log^2 n)$	$\mathcal{O}(n+m)$	PRAM CRCW
Klein	$\mathcal{O}(\log^2 n)$	$\mathcal{O}(\frac{n+m}{\log n})$	PRAM CRCW (randomized)