Tutorial on Graph Searching
Part1: Background and LBFS

Derek Corneil[1] and others to be named

[1]Computer Science, University of Toronto

Charles University
Sept. 30, 2013

Overview

BACKGROUND ON GRAPH SEARCHING

- Generic search, BFS, Lexicographic BFS (LBFS)
- LBFS and applications to various graph families

IMPLEMENTATION OF LBFS

- labels and partition refinement
- LBFS$^+$ and unit interval graph recognition

MAJOR THEOREMS REGARDING LBFS

APPLICATIONS OF LBFS

- Other early applications
- Recent applications

OPEN QUESTIONS and TOPICS FOR TOMORROW

BACKGROUND ON SEARCHING

- Searching a graph is a fundamental graph operation - throughout the talk we will assume our graphs are connected
- Visit every vertex and edge of $G(V, E)$
- Interested in the order vertices are visited
- BFS + DFS
    - Both described over a century ago for maze traversal
    - Many applications for both introduced in the 1970s and earlier
- Generic Search:
    - Introduced by Tarjan in 1970s
    - At each stage, if possible, visit an unvisited vertex that is adjacent to some previously visited vertex
- Graph searches have the following common structure:

Search Algorithm Outline:

Input: Graph $G(V, E)$
Output: Vector σ where $\sigma(i)$ is the i'th vertex chosen

```
1 $V' \leftarrow V$ {$V'$ is the set of unchosen vertices}
2 Step 1: Initialize data structure $D$ {$D$ will store a subset of $V'$ including $S$, the set of vertices eligible to be chosen at each step of the search}
3 for $i = 1$ to $n$ do
4     Step 2: Specify $S$ and choose $v$ from $S$ {"tiebreaking" if $|S| > 1$}
5     $\sigma(i) \leftarrow v$
6     $V' \leftarrow V' - \{v\}$
7     Step 3: Update $D$
8 end for
```

Generic Search:

Input: Graph $G(V, E)$
Output: Vector σ where $\sigma(i)$ is the i'th vertex chosen

```
1 $V' \leftarrow V$ {$V'$ is the set of unchosen vertices}
2 Step 1: $D$ is a set initialized to specific vertex $x$
3 for $i = 1$ to $n$ do
4     Step 2: $v$ is chosen from $S \leftarrow D$
5      $\sigma(i) \leftarrow v$
6      $V' \leftarrow V' - \{v\}$
7     Step 3: $D \leftarrow (D - \{v\}) \cup (N(v) \cap V')$
8 end for
```

Breadth First Search (BFS):

Input: Graph $G(V, E)$
Output: Vector σ where $\sigma(i)$ is the i'th vertex chosen

```
1 $V' \leftarrow V$ {$V'$ is the set of unchosen vertices}
2 Step 1: $D$ is a queue initialized to specific vertex $x$
3 for $i = 1$ to $n$ do
4     Step 2: $v \leftarrow S \leftarrow dequeue(D)$
5     $\sigma(i) \leftarrow v$
6     $V' \leftarrow V' - \{v\}$
7     Step 3: $enqueue(N(v) \cap V') \setminus D)$ on $D$
8 end for
```

Lexicographic BFS (LBFS)

In 1976 Rose, Tarjan and Lueker introduced LBFS and proved:
G is chordal iff an arbitrary LBFS is a perfect elimination ordering (PEO).
They also showed that other searches also have this property, namely Maximum Cardinality Search.

DEFINITIONS:

chordal: G is chordal iff there are no induced cycles of size greater than 3. (An alternate definition of G being chordal is that G has a Perfect Elimination Order (PEO).)

PEO: An ordering of $V: v_1, v_2, \cdots, v_i, \cdots v_n$ such that the neighbourhood of v_i in $G_i = G[v_1, v_2, \cdots, v_i]$ is a clique, $\forall i, 1 < i \leq n$. (i.e., v_i is *simplicial* in G_i).

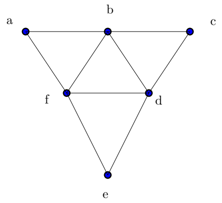
Roughly speaking, LBFS is a BFS procedure where "ties" are broken to favour vertices with earlier visited neighbours. Note: My orderings are Left to Right - often see Right to Left in the literature.

Lexicographic Breadth First Search (LBFS):

Input: Graph $G(V, E)$
Output: Vector σ where $\sigma(i)$ is the i'th vertex chosen

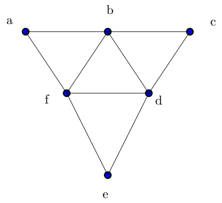
```
1 $V' \leftarrow V$ {$V'$ is the set of unchosen vertices}
2 Step 1: $D$ is a set of vertices, each with a label consisting of a string of decreasing positive integers; initially $D$ contains all vertices in $V'$, each with label $\epsilon$
3 for $i = 1$ to $n$ do
4     Step 2: $v$ chosen from $S \leftarrow$ the set of vertices of $D$ with lexicographically largest label
5     $\sigma(i) \leftarrow v$
6     $V' \leftarrow V' - \{v\}$
7     Step 3: $D \leftarrow D - \{v\}$, and append label $n - i + 1$ to the label of all vertices in $N(v) \cap V'$
8 end for
```

Example of LBFS



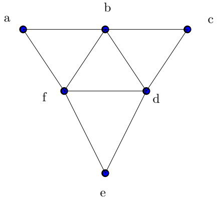
v	a	b	c	d	e	f
e	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ

Example of LBFS



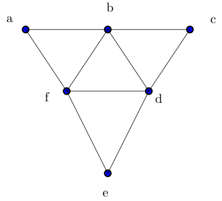
v	a	b	c	d	e	f
e	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ
f	ϵ	ϵ	ϵ	6		6

Example of LBFS



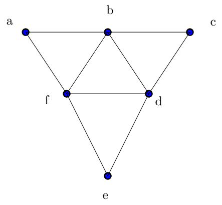
v	a	b	c	d	e	f
e	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ
f	ϵ	ϵ	ϵ	6		6
d	5	5	ϵ	65		

Example of LBFS



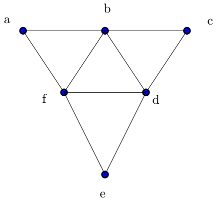
v	a	b	c	d	e	f
e	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ
f	ϵ	ϵ	ϵ	6		6
d	5	5	ϵ	65		
b	5	54	4			

Example of LBFS



v	a	b	c	d	e	f
e	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ
f	ϵ	ϵ	ϵ	6		6
d	5	5	ϵ	65		
b	5	54	4			
a	53		43			

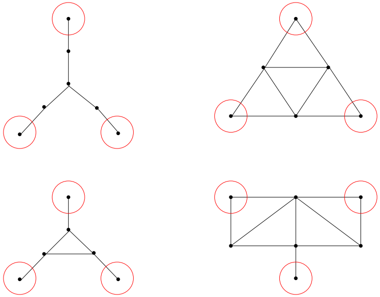
Example of LBFS

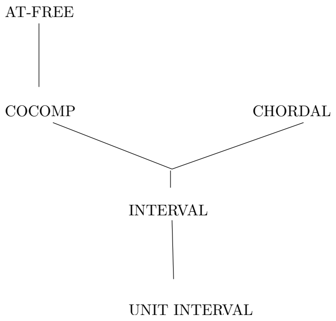


v	a	b	c	d	e	f
e	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ
f	ϵ	ϵ	ϵ	6		6
d	5	5	ϵ	65		
b	5	54	4			
a	53		43			
c			43			

LBFS and AT-free Graphs

- DEFINITION: An Asteroidal Triple (AT) is an independent triple of vertices, such that between any two there is a path that avoids the neighbourhood of the third.
- DEFINITION: $G(V, E)$ is AT-free if it has no AT.





Note: AT-free $\cap$ Chordal $=$ Cocomp $\cap$ Chordal $=$ Interval

Example: Dominating Pair in AT-free Graphs

Connected AT-free graphs have the beautiful property that they have a Dominating Pair, namely a pair of vertices $\{x, y\}$ such that *every* x, y path P dominates the graph (i.e., every vertex not on P has a neighbour that is on P).

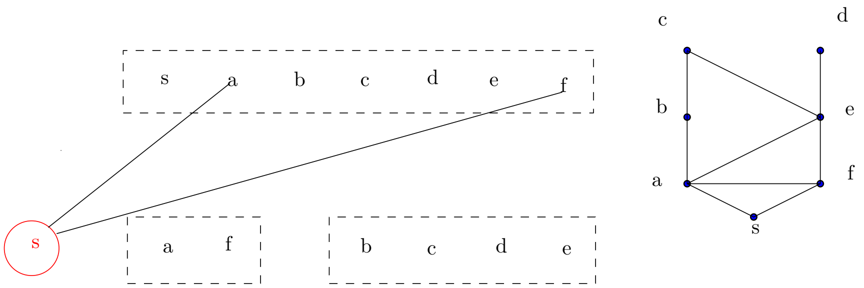
Example: Dominating Pair in AT-free Graphs

Connected AT-free graphs have the beautiful property that they have a Dominating Pair, namely a pair of vertices $\{x, y\}$ such that *every* x, y path P dominates the graph (i.e., every vertex not on P has a neighbour that is on P). Finding a Dominating Pair: C, Olariu and Stewart

1. Let σ be an arbitrary LBFS of G that ends at x
2. Let τ be an arbitrary LBFS of G that starts at x and ends at y
3. **return** $\{x, y\}$

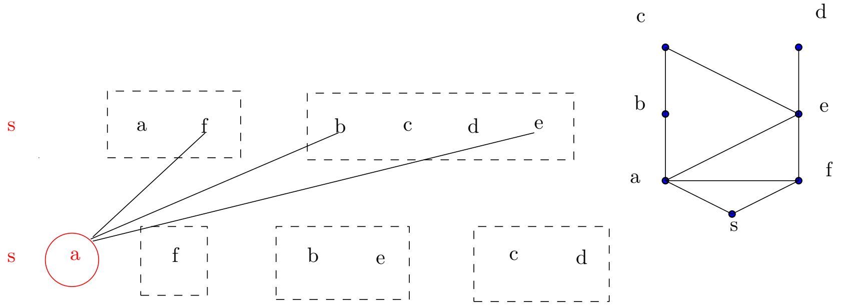
This was the first application of LBFS outside the chordal graph family.

Refinement Implementation of LBFS - Habib, McConnell, Paul + Viennot



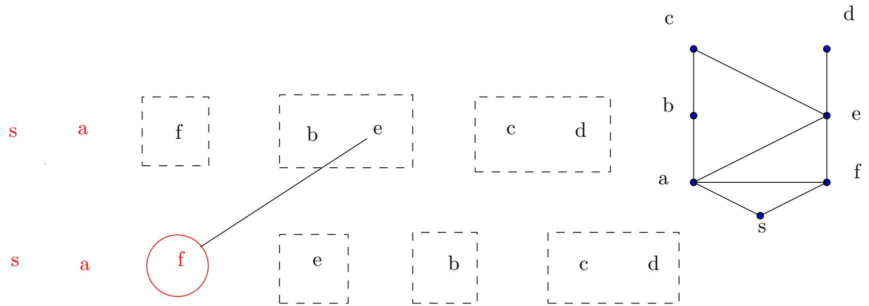
Refine each cell so that vertices adjacent to the pivot precede non-adjacent vertices. If there is more than one vertex in the leftmost cell, then choose the first one as the next pivot.

Refinement Implementation of LBFS - Habib, McConnell, Paul + Viennot



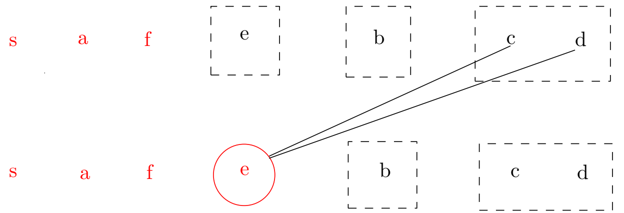
Refine each cell so that vertices adjacent to the pivot precede non-adjacent vertices. If there is more than one vertex in the leftmost cell, then choose the first one as the next pivot.

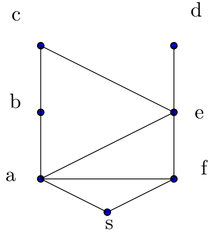
Refinement Implementation of LBFS - Habib, McConnell, Paul + Viennot



Refine each cell so that vertices adjacent to the pivot precede non-adjacent vertices. If there is more than one vertex in the leftmost cell, then choose the first one as the next pivot.

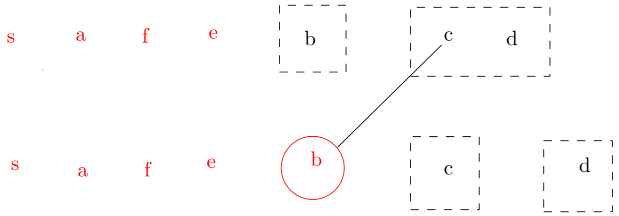
Refinement Implementation of LBFS - Habib, McConnell, Paul + Viennot

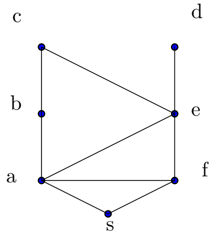




Refine each cell so that vertices adjacent to the pivot precede non-adjacent vertices. If there is more than one vertex in the leftmost cell, then choose the first one as the next pivot.

Refinement Implementation of LBFS - Habib, McConnell, Paul + Viennot





The resulting LBFS is: s a f e b c d.

Advantages of this implementation

- To do an LBFS of $\overline{G}$, just put non-adjacent vertices before adjacent vertices. Note that this produces an LBFS of $\overline{G}$ in time that is linear in the size of G, not $\overline{G}$, i.e., you don't have to calculate $\overline{G}$.

Advantages of this implementation

- To do an LBFS of $\overline{G}$, just put non-adjacent vertices before adjacent vertices. Note that this produces an LBFS of $\overline{G}$ in time that is linear in the size of G, not $\overline{G}$, i.e., you don't have to calculate $\overline{G}$.
- How to break ties? Suppose that S (called a slice) is the leftmost cell in the refinement where $|S| > 1$. In multi-sweep LBFS algorithms, we often want the pivot to be the LAST vertex in the previous LBFS search. Such an LBFS is called LBFS^{+}. To achieve this, simply reverse the order of the previous LBFS and use the standard implementation with this reversed order.

Example of LBFS$^+$

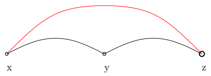
- DEFINITION: $G(V, E)$ is an interval graph if it is the intersection graph of subpaths of a path; namely, each vertex represents a subpath and two vertices are adjacent iff their subpaths intersect.

Example of LBFS$^+$

- DEFINITION: $G(V, E)$ is an interval graph if it is the intersection graph of subpaths of a path; namely, each vertex represents a subpath and two vertices are adjacent iff their subpaths intersect.
- DEFINITION: $G(V, E)$ is a unit interval graph if it is an interval graph where all intervals are of the same length (also known as proper interval graphs).

Example of LBFS$^+$

- DEFINITION: $G(V, E)$ is an interval graph if it is the intersection graph of subpaths of a path; namely, each vertex represents a subpath and two vertices are adjacent iff their subpaths intersect.
- DEFINITION: $G(V, E)$ is a unit interval graph if it is an interval graph where all intervals are of the same length (also known as proper interval graphs).
- THEOREM: G is unit interval iff there is an ordering of V such that for all $x < y < z, xz \in E$ implies $xy \in E$ AND $yz \in E$ (UI ORDER)



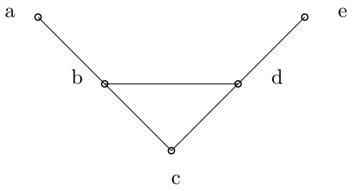
Recognition of Unit Interval Graphs [D.C. 2004]

1. Let σ be an arbitrary LBFS of G
2. $\sigma^+ \leftarrow \mathit{LBFS}^+(\sigma)$
3. $\sigma^{++} \leftarrow \mathit{LBFS}^+(\sigma^+)$
4. **if** σ^{++} is a UI ORDER **then** "G is unit interval" **else** "G is not unit interval"

$\sigma = c\ [b\ d]\ a\ e$

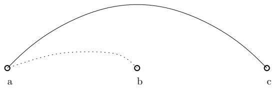
$\sigma^+ = e\ d\ [b\ c]\ a$

$\sigma^{++} = a\ b\ [c\ d]\ e$



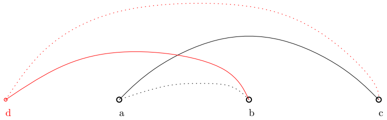
Major Theorems regarding LBFS

THEOREM (Golumbic; Dragan, Nicolai + Brandstadt):
An ordering σ is an LBFS ordering iff for all $a <_\sigma b <_\sigma c$ where $ac \in E$, $ab \notin E$, there exists $d <_\sigma a$ such that $db \in E$, $dc \notin E$.



Major Theorems regarding LBFS

THEOREM (Golumbic; Dragan, Nicolai + Brandstadt):
An ordering σ is an LBFS ordering iff for all $a <_\sigma b <_\sigma c$ where $ac \in E$, $ab \notin E$, there exists $d <_\sigma a$ such that $db \in E, dc \notin E$.



This property is crucial in the proofs of correctness of multi-sweep LBFS algorithms.

Major Theorems regarding LBFS - contd.

THEOREM (Chordal LBFS Lemma (C.O.S.)
Consider an arbitrary chordal graph G and two arbitrary LBFSs of G, σ and τ. If S is a slice in σ, then τ restricted to the vertices of S is an LBFS of the induced subgraph of G on S.

Major Theorems regarding LBFS - contd.

THEOREM (Chordal LBFS Lemma (C.O.S.))
Consider an arbitrary chordal graph G and two arbitrary LBFSs of G, σ and τ. If S is a slice in σ, then τ restricted to the vertices of S is an LBFS of the induced subgraph of G on S.

Note that this property doesn't even hold for cographs.

Major Theorems regarding LBFS - contd.

Notation: Given an LBFS ordering σ and two vertices, $u <_\sigma v$, we let $\Gamma^\sigma_{u,v}$ denote the smallest slice of σ containing both u and v.

Major Theorems regarding LBFS - contd.

Notation: Given an LBFS ordering σ and two vertices, $u <_\sigma v$, we let $\Gamma^\sigma_{u,v}$ denote the smallest slice of σ containing both u and v.

THEOREM: (The prior path Lemma (C.O.S.))
Let σ be an arbitrary LBFS of a graph G. Let t be the first vertex of the connected component of $\Gamma^\sigma_{u,v}$ containing u. Then there exists a t, u-path in $\Gamma^\sigma_{u,v}$ all of whose vertices, with the possible exception of u, are nonadjacent to v. Moreover, all vertices on this path, other than u, occur before u in σ. (Such a path is called a "prior path".)

Definitions

Vertex x is **simplicial** in graph G iff the neighbourhood of x in G is a clique.

Vertex x is **admissible** in graph G iff in G there does not exist vertices y, z with $x - y$ path P and $x - z$ path Q where $N(z) \cap P = \emptyset$ and $N(y) \cap Q = \emptyset$.

Major Theorems regarding LBFS - contd.

THEOREM (R.T.L.): If G is chordal, then any LBFS of G is a Perfect Elimination Order (PEO) (i.e., every vertex is simplicial in the subgraph induced on it and all vertices to its left in the LBFS ordering).

THEOREM (Fulkerson + Gross): G is chordal iff it has a PEO.

Major Theorems regarding LBFS - contd.

THEOREM (R.T.L.) If G is chordal, then any LBFS of G is a Perfect Elimination Order (PEO). (i.e., every vertex is simplicial in the subgraph induced on it and all vertices to its left in the LBFS ordering).

THEOREM (Fulkerson + Gross): G is chordal iff it has a PEO.

THEOREM (C.O.S.): If G is AT-free, then any LBFS of G is an Admissible Elimination Order (AEO). (i.e., every vertex is admissible in the subgraph induced on it and all vertices to its left in the LBFS ordering).

Unfortunately, G having an AEO does not imply that G is AT-free. But

THEOREM (C.+ Koehler): G is AT-free iff **ALL** LBFSs are AEOs.

Applications of LBFS

Early Applications:

- Diameter Estimation: In some graph families, the last vertex of an LBFS sweep is of high eccentricity.
- Powers of Graphs: Under some conditions, an LBFS of a given graph is a PEO of some powers of the given graph. Note that the graphs of these powers do not have to be calculated.
- Dominating Pair in AT-free Graphs: As we've seen.
- Recognition of Restricted Graph Families: As we've already seen, chordal graphs, and unit interval graphs - also distance hereditary (considered later) and some results on interval graph recognition (also discussed later).

Recognition of various restricted families of graphs

The general form of these algorithms is:

1. **do** a series of LBFS algorithms ending with a ordering τ of the vertices of G.
2. test τ to see if it satisfies a specific property.
3. **if** so **then** accept G **else** reject G.

What about interval graphs?

THEOREM (Raychaudhuri and others): G is an interval graph iff there is an ordering of V such that for every triple of vertices $\{u, v, w\}$ where $u < v < w$, and $uw \in E$ then $uv \in E$.

What about interval graphs?

THEOREM (Raychaudhuri and others): G is an interval graph iff there is an ordering of V such that for every triple of vertices $\{u, v, w\}$ where $u < v < w$, and $uw \in E$ then $uv \in E$.

Independently various groups have conjectured that the following algorithm recognizes interval graphs:

1. Let σ be an arbitrary LBFS of G
2. $\sigma^{+} \leftarrow LBFS^{+}(\sigma)$
3. $\sigma^{++} \leftarrow LBFS^{+}(\sigma^{+})$
4. $\sigma^{+++} \leftarrow LBFS^{+}(\sigma^{+}+)$
5. **if** σ^{+++} is an I ORDER **then** "G is interval" **else** "G is not interval"

What about interval graphs?

THEOREM (Raychaudhuri and others): G is an interval graph iff there is an ordering of V such that for every triple of vertices $\{u, v, w\}$ where $u < v < w$, and $uw \in E$ then $uv \in E$.

Independently various groups have conjectured that the following algorithm recognizes interval graphs:

1. Let σ be an arbitrary LBFS of G
2. $\sigma^{+} \leftarrow LBFS^{+}(\sigma)$
3. $\sigma^{++} \leftarrow LBFS^{+}(\sigma^{+})$
4. $\sigma^{+++} \leftarrow LBFS^{+}(\sigma^{++})$
5. **if** σ^{+++} is an I ORDER **then** "G is interval" **else** "G is not interval"

BUT Ma showed that for all positive integers c, there exists an interval graph G_c and an initial LBFS σ such that looping c times on LBFS^{+} will fail the I ORDER test!

Recognition of Interval Graphs [C.O.S.]

1. Let σ be an arbitrary LBFS of G
2. $\sigma^+ \leftarrow LBFS^+(\sigma)$
3. $\sigma^{++} \leftarrow LBFS^+(\sigma^+)$
4. $\sigma^{+++} \leftarrow LBFS^+(\sigma^{++})$
5. $\sigma^{++++} \leftarrow LBFS^+(\sigma^{+++})$
6. $\sigma^* \leftarrow LBFS^*(\sigma^{+++}, \sigma^{++++})$
7. **if** σ^* is an I ORDER **then** "G is interval" **else** "G is not interval"

To choose the first vertex of slice S, $LBFS^*$ chooses between α (the last S vertex of σ^{+++} and β (the last S vertex of σ^{+++}). There is a simple linear time implementation of $LBFS^*$.

Other families of graphs that have this type of LBFS recognition algorithm

Cographs: Precisely the graphs formed as follows:

- A vertex is a cograph
- If G_1 and G_2 are cographs, so is $G_1 \cup G_2$
- If G is a cogaph, so is $\overline{G}$

Facts about cographs:

- Cographs can be represented by a **cotree** where the leaves are the vertices of G and the non leaf nodes are either a "0-node" representing disjoint union of the children or a "1-node" representing join of the children (i.e., all edges with the endpoints in different children).
- G is a cograph iff G has no induced P_4.
- If T is the cotree of cograph G, then the cotree of $\overline{G}$ is $\overline{T}$ where the 0-nodes of T are replaced by 1-nodes and the 1-nodes of T are replaced by 0-nodes.

Cographs Cont.

The algorithm (Bretscher, C.H.P.) uses two LBFS sweeps, an arbitrary LBFS σ followed by an LBFS$^-$ which is performed on $\overline{G}$ and ties are broken by choosing the **earliest** S-vertex in σ.

In her Ph.D. thesis Bretscher showed that the same technique can be applied to achieve easy linear time recognition algorithms for:

- *P_4-reducible* graphs: every vertex appears in at most one P_4.
- *P_4-sparse* graphs: any set of 5 vertices induces at most one P_4.
- *distance hereditary* graphs: the distance between any two connected vertices of an induced subgraph equals their distance in the original graph. NOTE: for any vertex x and integer k, the set of vertices at distance k from x induces a cograph.

Applications of LBFS - Contd.

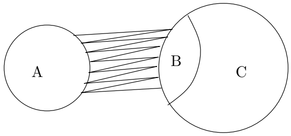
Recent Applications:

- Modular Decomposition
- Using LBFS as an ordering for an iterative algorithm
  - Split Decomposition
  - Circle Graph Recognition

Former Ph.D. student, Marc Tedder, is the key person in this work.

Modular Decomposition (T.C.H.P.)

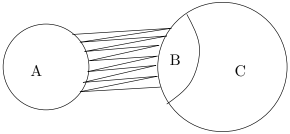
- DEFINITION: A strict subset A of V is a *module* if every vertex in $V \setminus A$ is either adjacent to all vertices of A or to no vertices of A.



A is a module; all $A * B$ edges are present and no $A * C$ edges are present. Note that a module is the result of replacing a vertex x where $N(x) = B$ by a graph A.

Modular Decomposition (T.C.H.P.)

- DEFINITION: A strict subset A of V is a *module* if every vertex in $V \setminus A$ is either adjacent to all vertices of A or to no vertices of A.

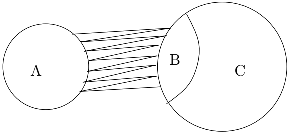


A is a module; all $A * B$ edges are present and no $A * C$ edges are present. Note that a module is the result of replacing a vertex x where $N(x) = B$ by a graph A.

  - Modular Decomposition is the reverse operation for every nontrivial module - this is represented by the MD-tree.

Modular Decomposition (T.C.H.P.)

- DEFINITION: A strict subset A of V is a *module* if every vertex in $V \setminus A$ is either adjacent to all vertices of A or to no vertices of A.



A is a module; all $A * B$ edges are present and no $A * C$ edges are present. Note that a module is the result of replacing a vertex x where $N(x) = B$ by a graph A.

- Modular Decomposition is the reverse operation for every nontrivial module - this is represented by the MD-tree.
- For many problems, a Divide and Conquer approach works, where we solve the problem for every prime module and then using the MD-tree, knit the solutions together to solve the problem for the given graph G.

MD contd.

- Previously, complicated linear time MD algorithms were presented, using two different techniques.

MD contd.

- Previously, complicated linear time MD algorithms were presented, using two different techniques.
- Marc developed a new version of LBFS that “back refines” as it proceeds. The new algorithm is linear time and combines previous techniques of factorizing permutations and recursive construction of the MD-tree.

MD contd.

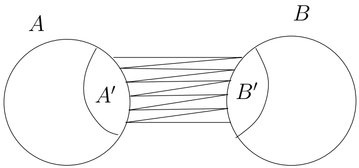
- Previously, complicated linear time MD algorithms were presented, using two different techniques.
- Marc developed a new version of LBFS that “back refines” as it proceeds. The new algorithm is linear time and combines previous techniques of factorizing permutations and recursive construction of the MD-tree.
- An earlier version of the algorithm has been implemented and users have stated that it is much simpler than previous algorithms.

Algorithm 2: Split Decomposition

- Split Decomposition (E. Gioan, P.T.C.):

  DEFINITION: A *split* of a connected graph $G = (V, E)$ is a bipartition (A, B) of V, where $|A|, |B| > 1$, such that

1. every vertex in $A' = N(B)$ is universal to $B' = N(A)$;
2. no other edges exist between vertices in A and B.

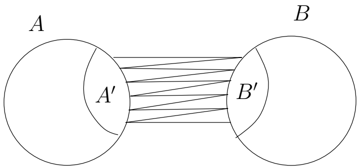


Algorithm 2: Split Decomposition

- Split Decomposition (E. Gioan, P.T.C.):

  DEFINITION: A *split* of a connected graph $G = (V, E)$ is a bipartition (A, B) of V, where $|A|, |B| > 1$, such that

1. every vertex in $A' = N(B)$ is universal to $B' = N(A)$;
2. no other edges exist between vertices in A and B.



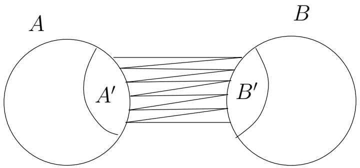
- The sets A' and B' are called the *frontiers* of the split.

Algorithm 2: Split Decomposition

- Split Decomposition (E. Gioan, P.T.C.):

  DEFINITION: A *split* of a connected graph $G = (V, E)$ is a bipartition (A, B) of V, where $|A|, |B| > 1$, such that

1. every vertex in $A' = N(B)$ is universal to $B' = N(A)$;
2. no other edges exist between vertices in A and B.



- The sets A' and B' are called the *frontiers* of the split.
- Split Decomposition (SD) is a generalization of MD ($A = A'$) and is also commonly used in a Divide and Conquer way, following the SD-tree.

Split Decomposition Contd.

- There is a linear time SD algorithm by Dahlhaus that is very hard to understand. A group in Paris has recently simplified this algorithm, but it is still difficult. Neither of these algorithms seems to generalize to circle graph recognition (our final example).

Split Decomposition Contd.

- There is a linear time SD algorithm by Dahlhaus that is very hard to understand. A group in Paris has recently simplified this algorithm, but it is still difficult. Neither of these algorithms seems to generalize to circle graph recognition (our final example).
- Our algorithm uses a LBFS preprocessing step and then incrementally updates the current SD-tree, according to this LBFS ordering.

Split Decomposition Contd.

- There is a linear time SD algorithm by Dahlhaus that is very hard to understand. A group in Paris has recently simplified this algorithm, but it is still difficult. Neither of these algorithms seems to generalize to circle graph recognition (our final example).
- Our algorithm uses a LBFS preprocessing step and then incrementally updates the current SD-tree, according to this LBFS ordering.
- The algorithm uses graph-labelled trees to represent the SD-tree, as proposed by E.G. and C.P. for distance hereditary graphs.

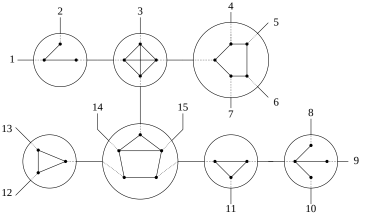
Split Decomposition Contd.

- There is a linear time SD algorithm by Dahlhaus that is very hard to understand. A group in Paris has recently simplified this algorithm, but it is still difficult. Neither of these algorithms seems to generalize to circle graph recognition (our final example).
- Our algorithm uses a LBFS preprocessing step and then incrementally updates the current SD-tree, according to this LBFS ordering.
- The algorithm uses graph-labelled trees to represent the SD-tree, as proposed by E.G. and C.P. for distance hereditary graphs.
- The new algorithm is straightforward, but is an inverse of Ackermann's function (α) off linear time (because of the use of Union-Find). Note that $\alpha \leq 4$ for all practical input.

Split Decomposition Contd.

A graph not containing a split is called *prime*.

DEFINITION: A graph-labelled tree (GLT) is a pair $(T, \mathcal{F})$, where T is a tree and $\mathcal{F}$ a set of graphs, such that each (nonleaf) node u of T is labelled by the graph $G(u) \in \mathcal{F}$, and there exists a bijection ρ_u between the edges of T incident to u and the vertices of $G(u)$. The vertices of such a $G(u)$ are called *marker vertices*. The leaves of T are the nodes of given graph $G(V, E)$.



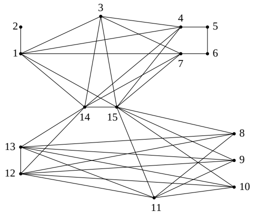
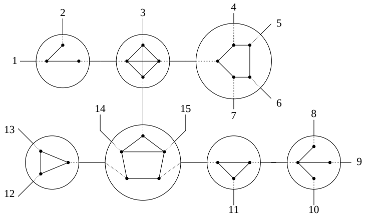
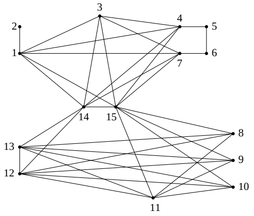


Figure: A graph-labelled tree $(T, \mathcal{F})$ and its accessibility graph $\mathcal{G}(T, \mathcal{F})$.

Split Decomposition Contd.





Note how the splits of *G* correspond to the node to node edges of the split tree as well as the splits of graph labels, when the graph label is not prime.

Cliques and stars are the only nonprime graph labels.

Split Decomposition Contd.

Algorithm Outline:

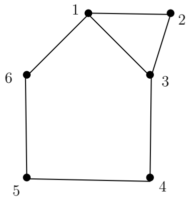
```
1  $ST(G_0) \leftarrow \emptyset$
2  Let $\sigma = x_1, x_2, \cdots, x_n$ be an arbitrary LBFS of $G$
3  **for** $i = 1$ **to** $n$ **do**
4        $ST(G_i) \leftarrow ST(G_{i-1}) + x_i$
5  **return** $ST(G_n)$
```

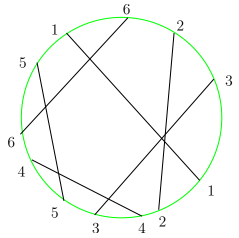
Note: The advantage of using the LBFS preprocessing step is that it allows a faster implementation than $O(n^3)$.

Algorithm 3: Circle Graph Recognition

- Circle Graph Recognition (G.P.T.C.): This is the first subquadratic algorithm for circle graph recognition and follows the same outline as the SD algorithm, with the same timing, i.e., inverse Ackermann's function off linear.

  DEFINITION: A circle graph is the intersection graph of chords of a circle.





Circle Graph Recognition Contd.

- Here the use of LBFS is based on a structural property of LBFS in circle graphs.

  THEOREM: Let G be a prime (with respect to SD) circle graph. If $x \in V(G)$ is the last vertex of an LBFS of G, then G has a chord diagram in which $N(x)$ is consecutive.
- No other known SD algorithm "lifts" to circle graph recognition in this way.

Open Questions and Topics for Tomorrow

1. Can LBFS be used for fast(er) recognition of:

Open Questions and Topics for Tomorrow

1. Can LBFS be used for fast(er) recognition of:
   - strongly chordal graphs [chordal and every cycle of even length at least 6 has an *odd chord* - i.e., the distance on the cycle between the endpoints is odd]

Open Questions and Topics for Tomorrow

1. Can LBFS be used for fast(er) recognition of:
   - strongly chordal graphs [chordal and every cycle of even length at least 6 has an *odd chord* - i.e., the distance on the cycle between the endpoints is odd]
   - chordal bipartite graphs [bipartite with no induced cycles of size greater than 4]

Open Questions and Topics for Tomorrow

1. Can LBFS be used for fast(er) recognition of:
    - strongly chordal graphs [chordal and every cycle of even length at least 6 has an *odd chord* - i.e., the distance on the cycle between the endpoints is odd]
    - chordal bipartite graphs [bipartite with no induced cycles of size greater than 4]
    - path graphs [the intersection graphs of subpaths of a tree]

Open Questions and Topics for Tomorrow

1. Can LBFS be used for fast(er) recognition of:
   - strongly chordal graphs [chordal and every cycle of even length at least 6 has an *odd chord* - i.e., the distance on the cycle between the endpoints is odd]
   - chordal bipartite graphs [bipartite with no induced cycles of size greater than 4]
   - path graphs [the intersection graphs of subpaths of a tree]
2. other applications of LBFS?

Open Questions and Topics for Tomorrow

1. Can LBFS be used for fast(er) recognition of:
   - strongly chordal graphs [chordal and every cycle of even length at least 6 has an *odd chord* - i.e., the distance on the cycle between the endpoints is odd]
   - chordal bipartite graphs [bipartite with no induced cycles of size greater than 4]
   - path graphs [the intersection graphs of subpaths of a tree]
2. other applications of LBFS?
3. Why is LBFS the only search that has a vertex ordering characterization?

Open Questions and Topics for Tomorrow

1. Can LBFS be used for fast(er) recognition of:
    - strongly chordal graphs [chordal and every cycle of even length at least 6 has an *odd chord* - i.e., the distance on the cycle between the endpoints is odd]
    - chordal bipartite graphs [bipartite with no induced cycles of size greater than 4]
    - path graphs [the intersection graphs of subpaths of a tree]
2. other applications of LBFS?
3. Why is LBFS the only search that has a vertex ordering characterization?
4. Are there good heuristic algorithms that could be built off graph searches?

Thank you for your attention