3-Coloring Circle Graphs in Quasipolynomial Time

with, Robert Ganian, Daniel Lokshtanov, Vaishali Surianarayanan

(SOSA 2026)

Ajaykrishnan E S
University of California, Santa Barbara

The 3-Coloring Problem

Proper 3-coloring of a graph G

Function $C : V(G) \rightarrow \{R, G, B\}$

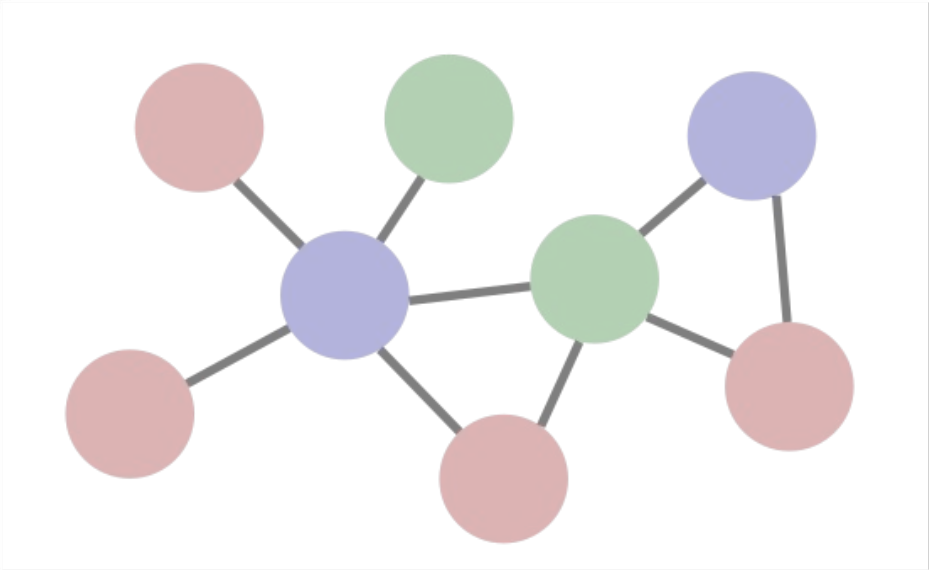
Such that, $C(u) \neq C(v)$ for every $uv \in E(G)$

The 3-Coloring Problem

Example:

Function
$C : V(G) \rightarrow \{R, G, B\}$

Such that,
$C(u) \neq C(v)$
for every $uv \in E(G)$

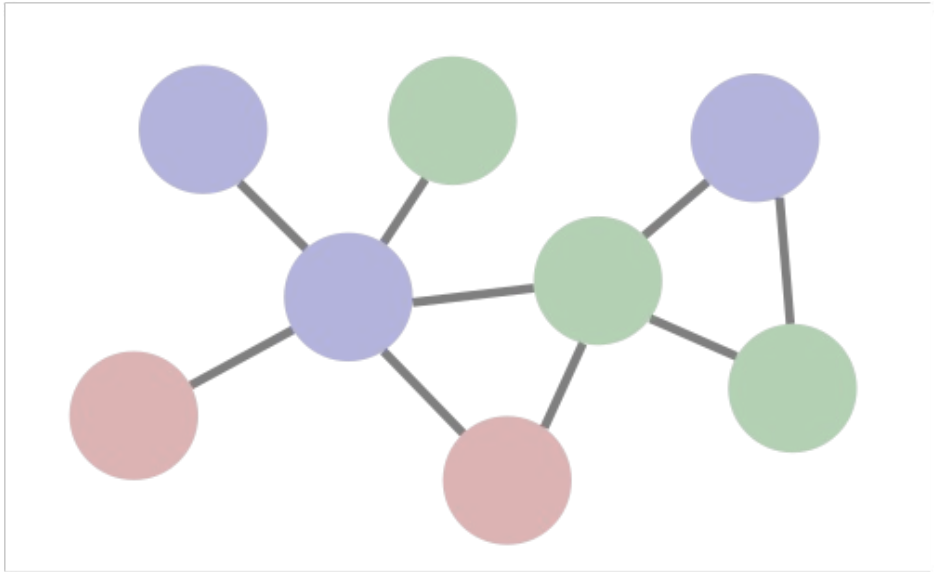


The 3-Coloring Problem

Non-Example:

Function
$C : V(G) \rightarrow \{R, G, B\}$

Such that,
$C(u) \neq C(v)$
for every $uv \in E(G)$



The 3-Coloring Problem

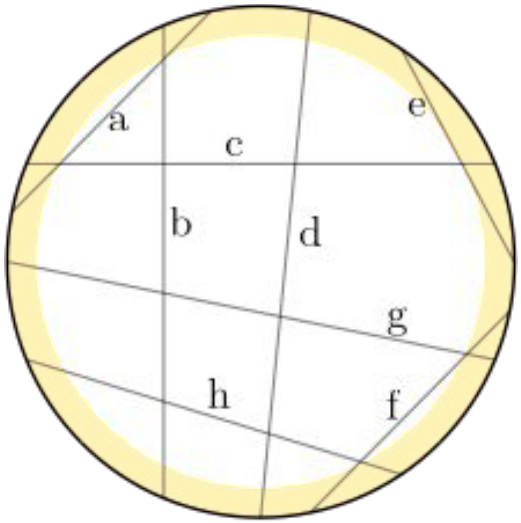
Input:

Graph G

Question:

Does G have a proper 3-coloring?

Circle Graph



Chord Diagram

Circle Graph

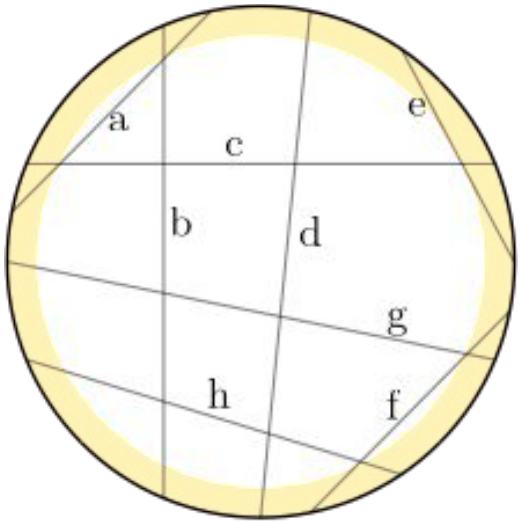
Vertices represent chords on a circle

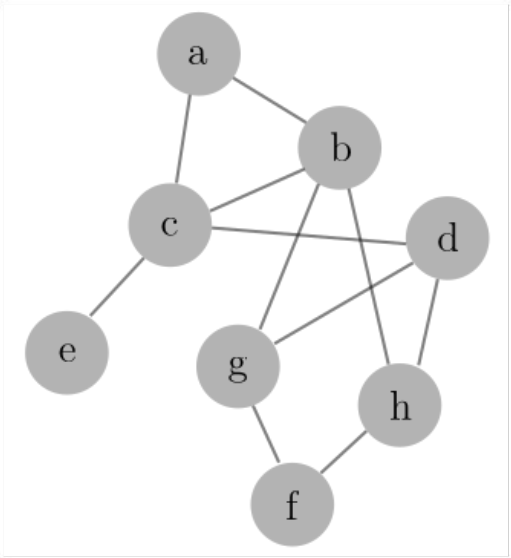
+

(x,y) is an edge if and only if chords x and y intersect

"Intersection graph of chords on a circle"

Circle Graph





3-Coloring Circle Graph

Input:

Chord Diagram
of a circle graph G

Question:

Does G have a
proper 3-coloring?

Open Problem

Does there exist a polynomial time algorithm for 3-coloring circle graphs?

Is it NP-complete?

Why 3-color Circle Graphs?

Connection to Book Embedding

Input: Graph G, integer k

Goal: Order the vertices of the graph and color the edges of the graph with k colors such that no two edges of same color cross

Why 3-color Circle Graphs?

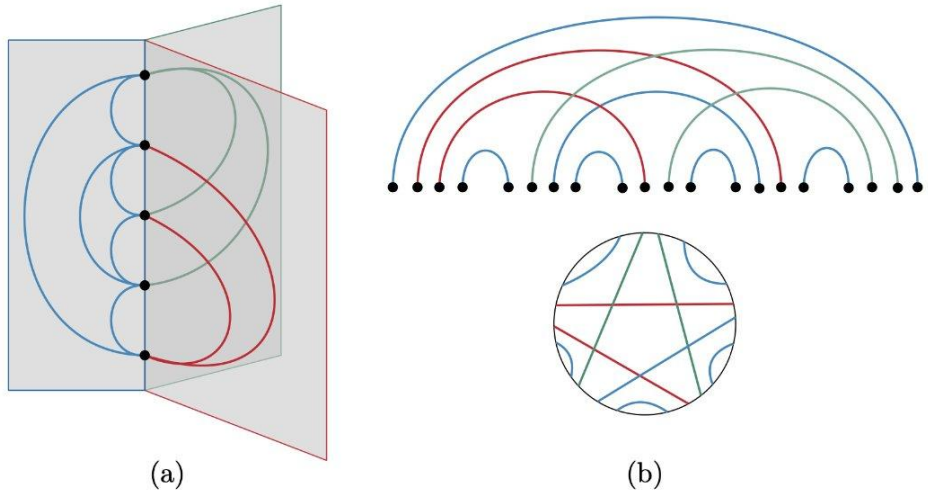


Fig. 1. (a) Book embedding for K_5. (b) Chord diagram of the corresponding circle graph H.

Motivations:

Graph Drawing

VLSI Design

Our Results

There exist a quasi-polynomial time algorithm for 3-coloring circle graphs.

$$n^{O(1)} < n^{O(\log n)} < 2^{O(n)}$$

Our Results

There exist a quasi-polynomial time algorithm for 3-coloring circle graphs.

Therefore it is not NP-complete*

If you agree that the statement “every problem in NP has a quasi-polynomial time algorithm” is *not* true in our reality

Algorithm

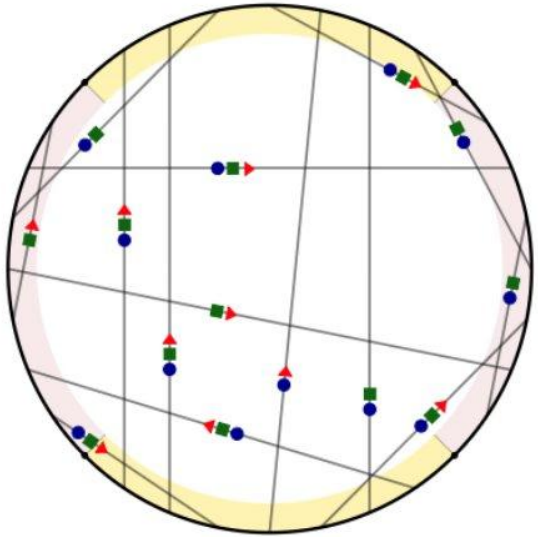
"List 3-coloring circle graphs"

Input:

Chord diagram

+

List of permitted colors for each chord



Algorithm - Overview

Recursive Algorithm

Recurse on n^a sub-instances, each of size n/b

Observe: Recursion depth is $O(\log n)$

Observe: Total runtime is $n^{O(\log n)}$

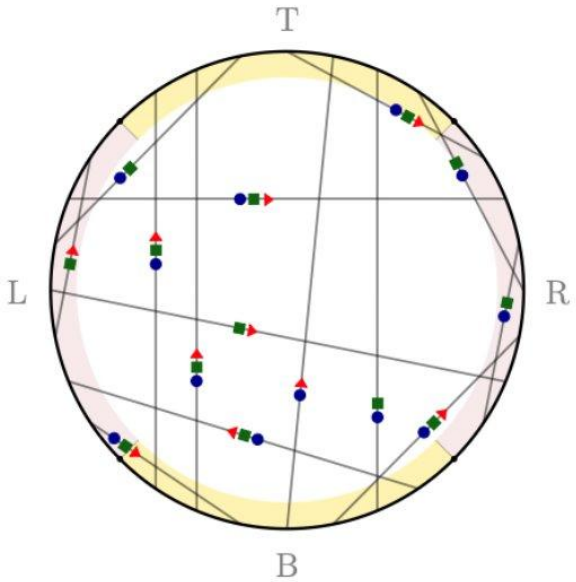
$\Longleftrightarrow$ $T(n) = n^a \cdot T(n/b)$

Algorithm - Step 1

- Partition the circle into four arcs (L,T,R,B) each containing ¼ fraction of the endpoints

n = 14 chords,

n/2 = 7 chords per arc



Algorithm - Step 1

- In a YES instance, one of the following is true.

All L-R chords are Red

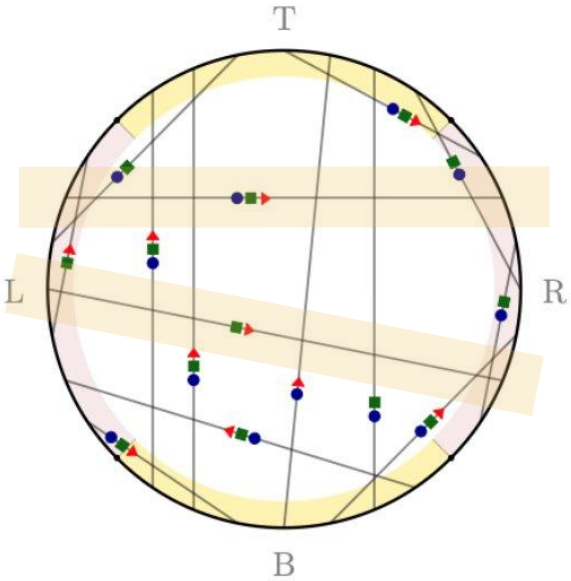
All L-R chords are Green

All L-R chords are Blue

All T-B chords are Red

All T-B chords are Green

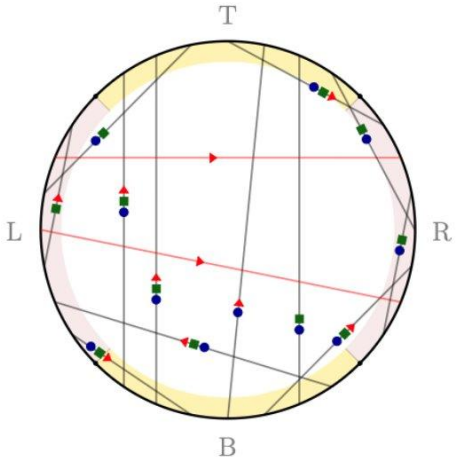
All T-B chords are Blue



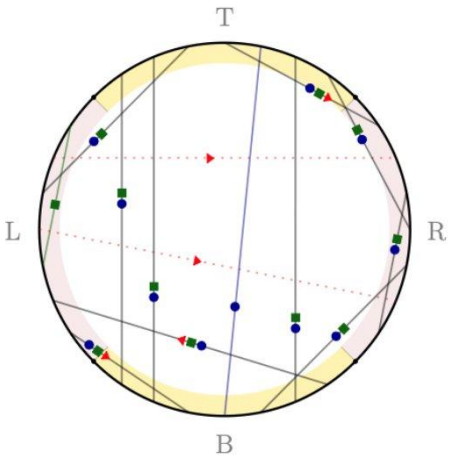
Algorithm – Step 1

- Partition the circle into four arcs (L,T,R,B) each containing ¼ fraction of the endpoints, and branch into six branches which either have no L-R chords or no T-B chords.

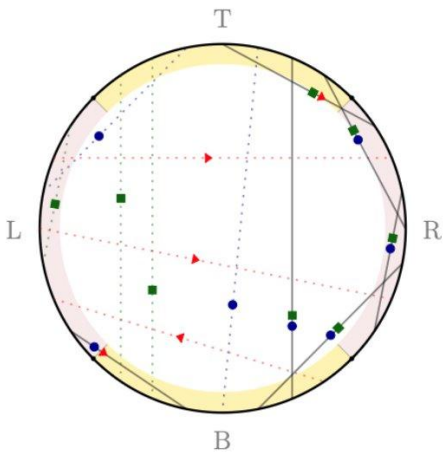
Algorithm - Step 1



Example Branch: All L-R chords are red.



Remove L-R chords and update



Update until every chord has at least two possible colors

Algorithm – Idea

(Assume we are in the branch with no L-R chords)

Maintain Invariant - “No L-R chords”

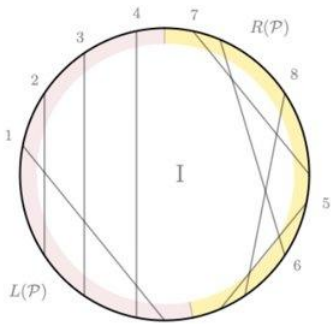
Attempt to “grow” L and R sides till T and B becomes empty

Algorithm - Idea

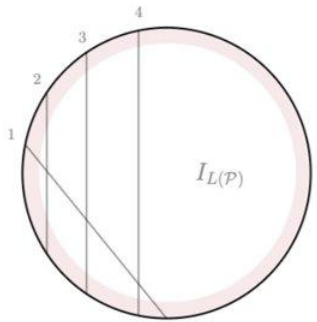
(Assume we are in the branch with no L-R chords)

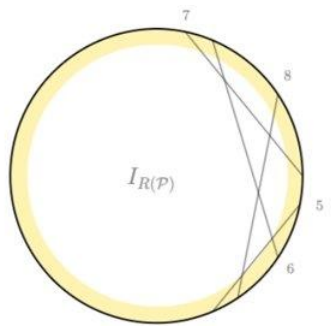
Maintain Invariant - “No L-R chords”

Attempt to “grow” L and R sides till T and B becomes empty



Old L, R are Grown





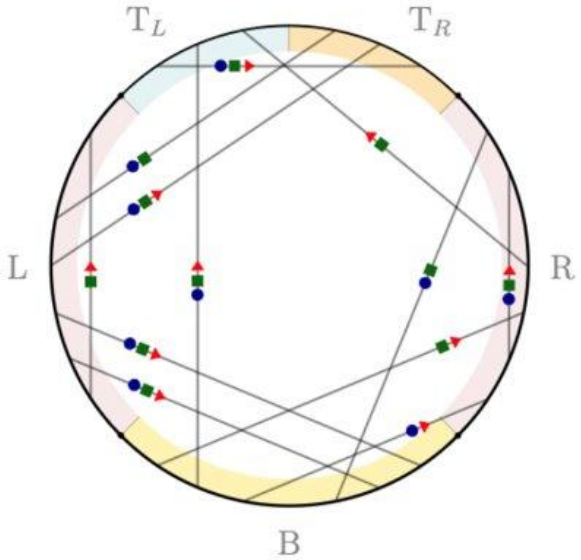
Instance is split into two separate ones

Old L, R contained ¼ fraction of the endpoints.

Thus new subinstance have at most 3n/4 vertices

Algorithm – Step 2

- As long as T contains some endpoint, partition it into T_L and T_R, each containing half the endpoints of T



Algorithm – Step 2

- In a YES instance, one of the following is true.

All L-T_R chords are Red

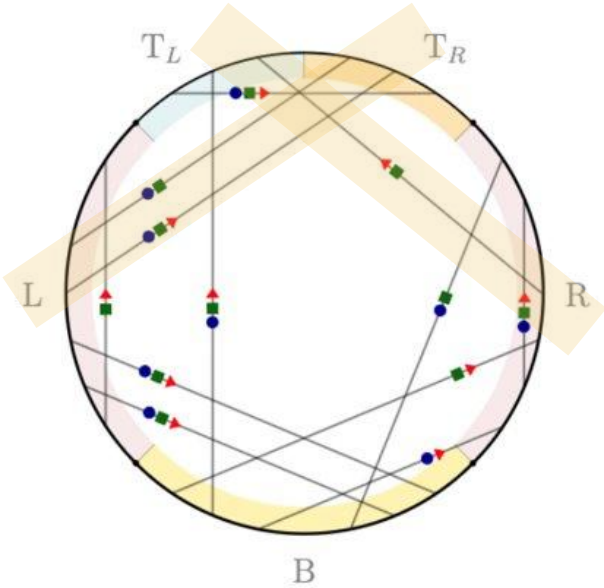
All L-T_R chords are Green

All L-T_R chords are Blue

All T_L-R chords are Red

All T_L-R chords are Green

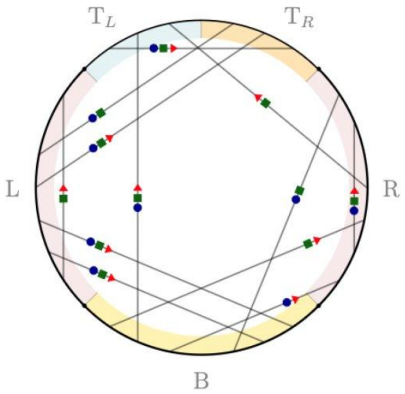
All T_L-R chords are Blue



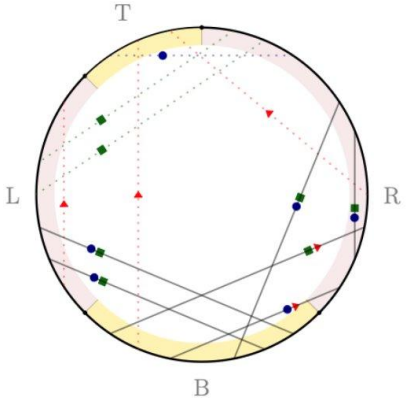
Algorithm – Step 2

- Again we get six branches, and in each one, either L or R is grown

(while maintaining the invariant)



Example Branch: All L-T_R chords are Green



After Updating, we grow R to include T_R

Algorithm – Step 2

- Keep repeating as long as T (and after that B) are non-empty.
- Have to repeat at most O(log n) times, since in each step we can grow by a constant fraction. Hence we get at most n^a sub-instances each of size n/b

Algorithm – Recap

- Divide the circle into (L,T,R,B) arcs
- Branch into six sub-instance with no L-R or T-B chords
- Grow L and R (or T and B) via branching to get at most n^a sub-instances each of size n/b
- Recurse

Thank You!