Matheus S. D'Andrea Alves

Recognition of (r, ℓ)-partite Graphs

2020

Abstract

The complexity to recognize if a graph has an (r, ℓ)-partition, i.e. if it can be partitioned into r cliques and ℓ independent sets, is well defined(1). However, as we will demonstrate, the literature-stabilished values for those on the P class can be improved. The following work provides a set of strategies and algortihms that pushes the previous results for the $(2, 1)$-partite (from n^4 to $n * m$), $(1, 2)$-partite (from n^4 to $n * m$) and $(2, 2)$-partite (from n^{12} to $n^2 * m$) recognition.
Keywords— (r, ℓ)-graphs, (r, ℓ)-partitions

List of Figures

List of Tables

Contents

1 Introduction

1.1 Current results

In this section we will explore the current state of the complexity analysis as r and ℓ grows. As we fullfill the following table we expose the strategies and how those can be used to enlight the more complex results.

The most trivial result is the recognition of the $(1, 0)$-graphs, as in order to recognize it we just need to know if $|E(G)| > 0$. Therefore it's complexity is $\mathcal{O}(1)$.

$r \setminus \ell$	0	1	2	3	4	$\dots$
0	-	?	?	?	?	$\dots$
1	$\mathcal{O}(1)$	?	?	?	?	$\dots$
2	?	?	?	?	?	$\dots$
3	?	?	?	?	?	$\dots$
4	?	?	?	?	?	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Table 1 – Incomplete complexity analysis of the (r, ℓ)-partite recognition problem

1.1.1 m-bounded results

Naturally, we wish to find the complexity for those problems of small partition cardinality.

At (2), König showed that it takes $\mathcal{O}(m)$ steps to recognize a bipartite graph. For complete graphs, is enough to check if $|E(G)| = n(n-1)/2$, if it doesn't then we have an answer; If it does, checking vertex by vertex if it's neighborhood contains all other vertex is $\mathcal{O}(m)$.

For co-bipartite graphs recognition, we need only to verify if it's complement is a bipartite graph. The recognition of a split graph can be done using their vertex degrees (**??**), obtaining all vertices degrees is $\mathcal{O}(m)$ therefore the recognition of split graphs is $\mathcal{O}(m)$

1.1.2 NP-Complete results

An adequate strategy at this moment is to find when the recognition problem gets NP-Complete.

At (3) is shown that 3-coloring a graph (i.e. assign a color between three possibles to each vertex such that no neighborhood repeats a color) is NP-Complete, it's trivial to see how the 3-coloring of a graph can be reduced to the problem of finding if a graph is a $(3,0)$-graph, therefore the recognition of $(3,0)$-graphs is NP-Complete.

It's noticible that the recognition of (r,ℓ)-graphs is monotonic, and therefore if the recognition of $(3,0)$-graphs are NP-Complete then the recognition of any $(r,0)$-graph or $(3,\ell)$-graph is NP-Complete for $r > 3$ and $\ell > 0$.

We can extrapolate these findings and argument that the recognition of a $(0,3)$-graph is also NP-Complete, as it is the same as recognize it's complement as a $(3,0)$-graph, and use the property of monotonicity to state that the recognition of any $(r,3)$-graph or $(0,\ell)$-graph is NP-Complete for $r > 0$ and $\ell > 3$.

$r \backslash \ell$	0	1	2	3	4	$\dots$
0	-	$\mathcal{O}(m)$	$\mathcal{O}(m)$	NPc	NPc	$\dots$
1	$\mathcal{O}(1)$	$\mathcal{O}(m)$	?	NPc	NPc	$\dots$
2	$\mathcal{O}(m)$	?	?	NPc	NPc	$\dots$
3	NPc	NPc	NPc	NPc	NPc	$\dots$
4	NPc	NPc	NPc	NPc	NPc	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Table 2 – Incomplete complexity analysis of the (r,ℓ)-partite recognition problem

1.1.3 Frontier results

Finally, the frontier cases $(1,2)$,$(2,1)$ and $(2,2)$ were subject of studies by Brandstädt(1, 4). He's findings show that:

- Recognition of $(1,2)$-graphs are $\mathcal{O}(n^4)$.
- Recognition of $(2,1)$-graphs are $\mathcal{O}(n^4)$.
- Recognition of $(2,2)$-graphs are $\mathcal{O}(n^{12})$.

$r \backslash \ell$	0	1	2	3	4	$\dots$
0	-	$\mathcal{O}(m)$	$\mathcal{O}(m)$	NPc	NPc	$\dots$
1	$\mathcal{O}(1)$	$\mathcal{O}(m)$	$\mathcal{O}(n^4)$	NPc	NPc	$\dots$
2	$\mathcal{O}(m)$	$\mathcal{O}(n^4)$	$\mathcal{O}(n^{12})$	NPc	NPc	$\dots$
3	NPc	NPc	NPc	NPc	NPc	$\dots$
4	NPc	NPc	NPc	NPc	NPc	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Table 3 – Current complexity analysis of the (r,ℓ)-partite recognition problem

1.2 Brandstädt's recognition of $(2, 1)$-graphs

In this section we will describe the algorithm designed by Brandstädt to recognize a $(2, 1)$-graph.

Let G be a graph. In order for G to accept a $(2, 1)$-partition a necessary condition is that for any vertex $v \in V(G)$:

- (N1) $N(v)$ induces a split graph.
- (N2) Or, $\bar{N}(v)$ should induces a bipartite graph

More specifically, if G has a $(2, 1)$-partition I_1, I_2, C then (N2) holds for every vertex $v \in C$ and (N1) holds for the vertices $v \in I_1 \cup I_2$

Let A be the set of vertices that is satisfied by (N1), let B be the set o vertices satisfied by (N2) then if $R = A \cap B = \emptyset$ then we already have a $(2, 1)$-partition of G. Otherwise

This conditions leads us to the following algorithm:

1: Get (2,1)-partition

```
def Get_21(Graph G) 2-1-Partition
  for v in V(G) do
    N = N(v) # open neighboorhood of v
    K = N̄(v) # V(G) - N
    if (!N.is_Split() and !K.is_Bipartite()) then
      return None; # G ∉ (2,1)
  end
  cl = { v in V(G) : !N(v).is_Split() and N̄(v).is_Bipartite() }
  bi = { v in V(G) : N(v).is_Split() and !N̄(v).is_Bipartite() }
  if ( !cl.is_Clique() or !bi.is_Bipartite()) then
    return None; # G ∉ (2,1)
  R = {v in V(G) : N(v).is_Split() and N̄(v).is_Bipartite()}
  if R.is_Empty() then
    return new 2-1-Partition {
      I_1: bi.I_1,
      I_2: bi.I_2,
      C: cl
    }
  else return AllocateR()
end
```

2: My Code

```
def Get_21(Graph G) 2-1-Partition
  for v in V(G) do # O(n)
      N = N(v) # open neighboorhood of v
      K = N̄(v) # V(G) - N
      if (!N.is_Split() and !K.is_Bipartite()) then
        return None; # G ∉ (2,1)
```

```
  end
  cl = { v in V(G) : !N(v).is_Split() and N̄(v).is_Bipartite() }
  bi = { v in V(G) : N(v).is_Split() and !N̄(v).is_Bipartite() }
  if ( !cl.is_Clique() or !bi.is_Bipartite()) then
    return None; # G ∉ (2,1)
  R = {v in V(G) : N(v).is_Split() and N̄(v).is_Bipartite()}
  if R.is_Empty() then
    return new 2-1-Partition {
      I_1: bi.I_1,
      I_2: bi.I_2,
      C: cl
    }
  else
    for v in R do # O(n²)
      H = new 3-1-Partition{
        I_1: N̄(v).I_1
        I_2: N̄(v).I_2
        I_3: N(v).I
        C: N(v).C
      }
      I = ⋃_{k=1}^{3} H.I_k
      capable = { u in I : |N(u) ∩ H.C| ≥ |H.C| − 2 }
      if !(I \ capable).is_Bipartite() then next;# This v of R can't induce a 2-1
      capable = { u in capable : }
end
```

Bibliography

1 BRANDSTÄDT, A. *Partitions of graphs into one or two independent sets and cliques*. 1984.

2 KÖNIG, D. *Theorie der endlichen und unendlichen Graphen: Kombinatorische Topologie der Streckenkomplexe*. [S.l.]: Akademische Verlagsgesellschaft mbh, 1936. v. 16.

3 GAREY, M. R.; JOHNSON, D. S. *Computers and intractability*. [S.l.]: wh freeman New York, 2002. v. 29.

4 BRANDSTÄDT, A. Partitions of graphs into one or two independent sets and cliques. *Discrete Mathematics*, Elsevier, v. 152, n. 1-3, p. 47–54, 1996.