International Journal of Mathematics and
Computer Science, **11**(2016), no. 1, 33–42

Edge Grundy Numbers of $P_3\Box P_n$ and $P_3\Box C_n$

Loren Anderson

Department of Mathematics
North Dakota State University
Fargo, ND 58102, USA

email: loren.james.anderson@ndsu.edu

(Received October 22, 2015 Accepted November 8, 2015)

Abstract

In [1], the quantities $\Gamma'(P_n\Box P_n)$ and $\Gamma'(P_n\Box C_n)$ were determined except for $\Gamma'(P_3\Box P_n), n \in \{5,6,7\}$ and $\Gamma'(P_3\Box C_n), n \in \{4,5,6\}$. Here, we state these numbers and explain the computer algorithm coded in Haskell used to find them.

1 Grundy Colorings

The graphs we study in this paper are *finite simple* graphs, meaning they have a finite number of vertices and edges with no multiple edges or loops. A *proper coloring* of a graph G is a coloring of the vertices such that adjacent vertices of G are colored differently. Here, we are concerned with *Grundy* colorings, proper colorings of the vertices of G with positive integers such that if $v \in V(G)$, is colored with $c > 1$, then all colors $1, ..., c-1$ appear on neighbors of v. We denote the *Grundy number* of G, the greatest number of colors appearing in a Grundy coloring of G, as $\Gamma(G)$. An *edge Grundy coloring* of a graph G is a Grundy coloring of the line graph of G, $L(G)$. We denote the edge Grundy number of G as $\Gamma(G)$; $\Gamma'(G) = \Gamma(L(G))$. Let P_n denote the path on n vertices and C_n the cycle on n vertices. The Cartesian product will be denoted as $\Box$. Here, $P_n\Box P_m$ is the n by m grid, and $P_n\Box C_m$ is the n by m cylindrical grid. In [1], the values of $\Gamma'(P_n\Box P_m)$ are determined

Key words and phrases: Proper coloring, Grundy coloring, Grundy number, line graph, path, cycle, grid, Cartesian product of graphs, Haskell code.
AMS (MOS) Subject Classifications: 05C15, 68R10.
ISSN 1814-0432, 2016, http://ijmcs.future-in-tech.net
This research was supported by NSF grant no. 1262930.

for all values of $n \leq m$ except for $(n,m) \in \{(3,m)|5 \leq m \leq 7\}$, and the values of $\Gamma'(P_n \Box C_m)$ are determined for all values of n and m except for $(n,m) \in \{(3,m)|4 \leq m \leq 6\}$. The aim of this paper is to fill these gaps left in [1]. Computer search will be employed; the following, from [1], together with some other observations, to be given later, permit simplification and shortening of the search.

A *full partial Grundy coloring* of G is an assignment of positive integers to some of the vertices of G so that adjacent vertices are assigned different integers, and if $v \in V(G)$ is assigned $c > 1$, then $1, ..., c-1$ appear on neighbors of v. The following is well known (see [1]).

Lemma 1.1. *Every full partial Grundy coloring of a graph G can be extended to a Grundy coloring of G.*

Corollary 1.1. *If H is an induced subgraph of G, then* $\Gamma(H) \leq \Gamma(G)$.

Corollary 1.2. *If* $\Gamma'(G) \leq m$ *and a full partial Grundy coloring of G is found which uses m colors, then* $\Gamma'(G) = m$.

2 Main Results

In [1], the following was shown:

$$6 \leq \Gamma^{'}(P_3 \Box P_m) \leq 7, m \in \{5,6,7\}$$

$$6 \leq \Gamma^{'}(P_3 \Box C_m) \leq 7, m \in \{4,5,6\}.$$

Using computer aid, we determined that the graphs G among the $P_3 \Box H$, above, such that $\Gamma'(G) = 7$ are as follows:

G	Number of edge Grundy colorings using 7 colors
$P_3 \Box P_5$	0
$P_3 \Box P_6$	0
$P_3 \Box P_7$	1280
$P_3 \Box C_4$	0
$P_3 \Box C_5$	320
$P_3 \Box C_6$	5100

Take note that the number of graphs is not reduced by automorphisms. We start by assigning numbers to vertices of the line graph and then cycle through all possible colorings of the vertices to determine when $\Gamma'(G) = 7$. In this fashion, each distinct graph counted in the table has unique set of vertex number/color pairs. For each line graph $L(G)$, we determined that only a single vertex is able to be colored 7 in order to achieve a Grundy coloring of $L(G)$.

Theorem 2.1.

$$\Gamma'(P_3\Box P_5) = \Gamma'(P_3\Box P_6) = \Gamma'(P_3\Box C_4) = 6$$

Theorem 2.2.

$$\Gamma'(P_3\Box P_7) = \Gamma'(P_3\Box C_5) = \Gamma'(P_3\Box C_6) = 7$$

3 $P_3\Box P_7$, $P_3\Box C_5$, $P_3\Box C_6$

In this section, we display examples of colorings that prove Theorem 2.2:

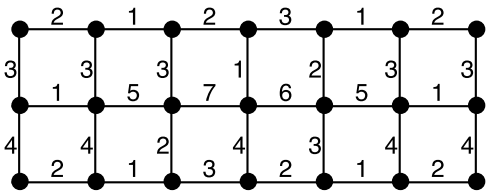

FIGURE 1. $P_3\Box P_7$

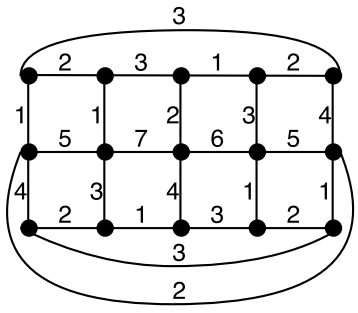

FIGURE 2. $P_3\Box C_5$

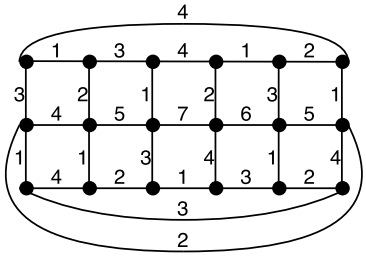

FIGURE 3. $P_3 \square C_6$

4 Algorithm for $P_3 \square P_5$, $P_3 \square P_6$, $P_3 \square C_4$

In this section, we describe the computer algorithm used to establish Theorem 2.1. The hurdle of even using a computer algorithm is that we simply cannot examine all the possibilities of colorings and then eliminate those that do not satisfy the prescribed conditions. To yield a Grundy coloring, each vertex must have color less than or equal to its degree plus one. The graph $L(P_3 \square P_5)$ below illustrates the degree of each vertex. The number on each edge is the degree of the corresponding vertex in $L(P_3 \square P_5)$. Multiplying together all possibilities for the color of each vertex yields: $4^8 \cdot 5^4 \cdot 6^8 \cdot 7^2 \geq 10^{15}$ possible colorings, which is too many to check - even for one of our smallest graphs. This mandates a more sophisticated approach.

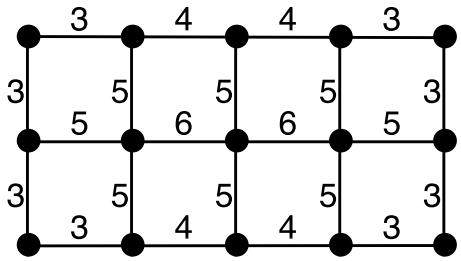

FIGURE 4. $P_3 \square P_5$

We start with a line graph that has one vertex colored 7; the choice of this vertex is determined by the symmetry of the graph. Then, we choose a set of uncolored vertices and assign all possible permutations of colorings to

the vertices. This yields a set of partially colored graphs. Then we throw out all graphs whose colorings are not proper. Next, we determine which newly colored vertices have all of their neighbors also colored. If all neighbors of any newly colored vertex are colored, we check the Grundy condition on the newly colored vertex. This allows us to throw out the graphs that are not Grundy (and hence will never be Grundy under coloring the remaining vertices). We then choose another set of uncolored vertices and start the whole process over with our set of partially colored graphs until all vertices are colored. If there is a graph remaining at the end of the program, its coloring must be a proper Grundy coloring, and we conclude that the edge Grundy number is 7 (because we starting with a graph having one vertex colored 7). The careful choice of sets of vertices is the main aspect of reducing the time complexity of this program.

Next, we explain the algorithm. First, define $S(v)$ as the set of uncolored neighbors of a vertex v and $T(v)$ as the set of colored neighbors. Furthermore, let $c(T(v))$ be the set of colors of the colored neighbors of v. We define $p1$ and $p2$, the respective proper coloring and Grundy conditions on a single vertex. For $v \in V(G)$ to satisfy $p1$, either v must be uncolored, or each neighbor of v must have a different color from the color of v. For $v \in V(G)$ to satisfy $p2$, where v is colored c, the following must be true:

$$|S(v)| \geq |\{1, 2, ..., c-1\} - c(T(v))|.$$

This condition is stating that the number of uncolored neighbors of v is at least the number of missing colors on neighbors of v needed for v to satisfy the Grundy condition. We present the algorithm generating all Grundy graphs with one vertex colored 7 below.

ALGORITHM
- color one vertex 7
- WHILE there exists an uncolored vertex v
- FOR ALL colors c at most one more than the degree of v
- color v c
- IF v satisfies $p1$ and $p2$
- IF colored neighbors of v satisfy $p2$

We now present a short proof of the validity of this algorithm by induction. When the first vertex is colored 7, it trivially satisfies $p1$ because there are no other colored vertices. Also, this vertex satisfies $p2$ if and only if the colored vertex has degree 6 or greater; otherwise it cannot be adjacent to vertices colored from 1 through 6. Next, assume that there are n vertices currently colored, all satisfying $p1$ and $p2$. Choose an uncolored vertex v and color it c. It suffices to check that only v satisfies $p1$ to be certain that all other n colored vertices will satisfy $p1$; v was the only new vertex colored. If v satisfies $p2$, it suffices to check the colored neighbors of v to see if they satisfy $p2$. This is because the set $c(T(u))$ for only neighbors u of v may change when v is colored. When all vertices of the graph are colored, each vertex must satisfy $p1$ and $p2$. Each vertex satisfying $p1$ is equivalent to the coloring being a proper coloring, and each vertex satisfying $p2$ along with $p1$ is equivalent to the coloring being a Grundy coloring.

All that is left to discuss is the strategy of choosing the vertex coloring order. This varies based on which graph we are analyzing. The order was chosen to check $p2$ often, as it seemed to be the more restrictive of the two conditions for these graphs after some trial and error while coding.

5 $P_3 \square C_4$

The main feature of $P_3 \square C_4$ is its symmetry. This cylindrical graph 'wraps around' itself and has symmetry of a cylinder. The only possible edge that can be colored 7 is one on the middle row, or one labeled 1, 4, 5, or 20 in figure 5. Therefore, we can assume a single edge on this row is colored 7 by the symmetry; here, we assume that the edge 1 is colored 7. Upon inplementing the computer program with respect to this graph, we determined that there were 0 possible Grundy colorings, so $\Gamma'(P_3 \square C_4) = 6$.

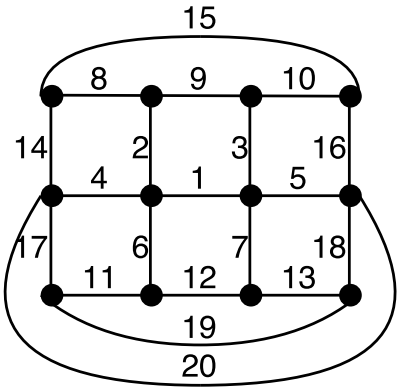

FIGURE 5. $P_3\Box C_4$

6 $P_3\Box P_5$ and $P_3\Box P_6$

The graph $P_3\Box P_6$ does not exhibit the same type of symmetry, but we can reduce the problem down to the two cases shown below where edge 1 is colored 7 in each case.

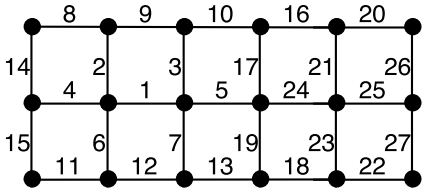

FIGURE 6. $P_3\Box P_6$

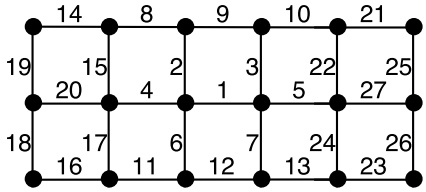

FIGURE 7. $P_3\Box P_6$

In either case, the computer program determined that there were 0 Grundy graphs. We conclude that $\Gamma'(P_3\Box P_6) = 6$. The $L(P_3\Box P_5)$ graph is an induced subgraph of $L(P_3\Box P_6)$. Therefore, $\Gamma'(P_3\Box P_5) \leq \Gamma'(P_3\Box P_6) = 6$ by Corollary 1.1. However, we know that $6 \leq \Gamma'(P_3\Box P_5)$ by the work of [1]. Therefore, $\Gamma'(P_3\Box P_5) = 6$.

7 Code

Below is the Haskell code used to implement the algorithm described in section 4. We display how it checks the graph $L(P_3 \square C_4)$ as described in section 5.

```
import qualified Data.Map as Map
import Data.List
type Node = (Int,([Int],Int))
type Graph = Map.Map Int ([Int],Int)
--gets degree of a vertex
degree :: Int -> Graph -> Int
degree x g = case Map.lookup x g of
               Just v -> length (fst v)
               Nothing -> 0
--gets color of a vertex
color :: Int -> Graph -> Int
color x g = case Map.lookup x g of
              Just v -> (snd v)
              Nothing -> error "Not good"
--determines if a vertex is colored
isColored :: Int-> Graph -> Bool
isColored x g = case Map.lookup x g of
                  Just v -> if (snd v)==0 then False else True
                  Nothing -> error "Not good"
--checks if vertex is colored differently from each member of set of vertices
checkProperColor :: Int -> [Int] -> Graph -> Bool
checkProperColor _ [] _ = True
checkProperColor y (x:xs) g = ((color y g) /= (color x g))
                              && checkProperColor y xs g
--changes color of vertex to specified new color
changeColor :: Int -> Int -> Graph -> Graph
changeColor y c g= Map.insertWith addColor y ([],c) g
--helper function for changeColor
addColor :: ([Int],Int) -> ([Int],Int) -> ([Int],Int)
```

```
addColor (xs,d) (zs,y) = (zs,d)
--gets neighbors of a certain vertex
getNeighbs :: Int -> Graph -> [Int]
getNeighbs x g = case Map.lookup x g of
                  Just v ->  (fst v)
                  Nothing -> error "Not good"
--determines if a set of vertices satisfies the Grundy condition
checkGrundyColor :: [Int] -> Graph -> Bool
checkGrundyColor [] _ = True
checkGrundyColor (y:ys) g | isColored y g =
               checkGrundy (getNeighbs y g) (color y g) g && checkGrundyColor ys g
                          | otherwise  =  checkGrundyColor ys g
--determines if a single vertex satisfies the Grundy condition
checkGrundy :: [Int] -> Int-> Graph -> Bool
checkGrundy xs c g = (notColored xs g) >=
     ((c-1)-(length(delete 0 (filter (<c) (nub (isOne xs g))))))
--determines the set of colors of vertices to which a single vertex is adjacent
isOne :: [Int] -> Graph -> [Int]
isOne [] g = []
isOne (x:xs) g =(color x g):(isOne xs g)
--determines how many vertices in a set are not colored yet
notColored ::[Int] -> Graph -> Int
notColored [] _ = 0
notColored (x:xs) g | not (isColored x g) =  1 + (notColored xs g)
                    | otherwise = notColored xs g
--Checks if a graph is a proper coloring and a Grundy coloring
goodGraph ::  [Int] -> Graph -> Bool
goodGraph (y:ys) g = checkGrundyColor (y:zs) g && checkProperColor y zs g
                        where zs = getNeighbs y g
--START HERE: input graph
grundy7 :: Graph -> Int
grundy7 g = cycleColors xs ((degree (head xs) g)+1) g
            where xs = Map.keys g
--cycles through colors of a new vertex
```

```
cycleColors :: [Int] -> Int -> Graph -> Int
cycleColors  _ 0 _ = 0
cycleColors [] _ _ = 0
cycleColors (y:ys) d g | y==1 = checkEverything (y:ys) (changeColor 1 7 g)
                       | otherwise =  (checkEverything (y:ys) (changeColor y d g))
                                                        + (cycleColors (y:ys) (d-1) g)
--Checks if a graph is proper, Grundy, and then if it is fully colored
checkEverything :: [Int] -> Graph -> Int
checkEverything (y:ys) g | goodGraph (y:ys) g = if (ys == []) then 1 else
                                      cycleColors ys ((degree (head ys) g)+1) g
                         | otherwise = 0
p3c4Node :: [Node]
p3c4Node = [(1,([2,3,4,5,6,7],0)),(2,([1,4,6,8,9],0)),(3,([1,5,7,9,10],0)),
(4,([1,2,6,14,17,20],0)),(5,([1,3,7,16,18,20],0)),(6,([1,2,4,11,12],0)),
(7,([1,3,5,12,13],0)),(8,([2,9,14,15],0)),(9,([2,3,8,10],0)),
(10,([3,9,15,16],0)),(11,([4,6,17,19],0)),(12,([6,7,11,13],0)),
(13,([7,12,18,19],0)),(14,([4,8,15,17,20],0)),(15,([8,10,14,16],0)),
(16,([5,10,15,18,20],0)),(17,([4,11,14,19,20],0)),(18,([5,13,16,19,20],0)),
(19,([11,13,17,18],0)),(20,([4,5,14,16,17,18],0))]
p3c4 :: Graph
p3c4 = Map.fromList p3c4Node
```

Acknowledgements. The author would like to thank Kei Davis for assistance in learning Haskell and suggestions for implementing the code. Also, the author would like to thank Dr. Peter Johnson of Auburn University for his suggestions and help throughout and after the 2013 Auburn Mathematics REU.

References

[1] Loren Anderson, Matthew DeVilbiss, Sarah Holliday, Peter Johnson, Anna Kite, Ryan Matzke, and Jessica McDonald, The edge Grundy numbers of some graphs, submitted to Utilitas Mathematica.