Algorithms for Constructing Zero-Divisor Graphs of Commutative Rings

Joan Krone

Abstract

The idea of associating a graph with the zero-divisors of a commutative ring was introduced in [3], where the author talked about colorings of such graphs. In [1], the authors considered a conjecture made by Beck [3] and found a counterexample to the conjecture. In [2], the authors investigated the interplay between the ring-theoretic properties of a ring and a graph associated with the ring, namely the zero-divisor graph.

Here we present algorithms for deriving zero-divisor graphs for commutative rings. These algorithms are recursive in nature and construct the graph for a given ring from sub-graphs which themselves are zero-divisor graphs of rings of smaller orders. We put forward algorithms to derive zero-divisor graphs for the following types of rings: Z_n (all integers with the usual addition and multiplication mod n), products of Z_n rings, and E_n (all even integers with the usual addition and multiplication mod n).

Introduction

Given a commutative ring, *R*, with unity and the set of associated zero divisors by *Z(R),* define the zero-divisor graph of *R*, *G(R)* as a graph whose vertices are the nonzero zero-divisors of *R* and whose edges are the joins of those vertices, *v1* and *v2* such that *v1*v2* = 0, i.e., *v1* and *v2* are adjacent iff *v1*v2* = 0. Note that *G(R)* is empty iff *R* is an integral domain. Sometimes the zero-divisor graph for *R* is allowed to have 0 as a vertex, in which case 0 has an edge to every other node in the graph. For simplicity, we are choosing the definition that does not include the 0 as a vertex, since putting it in simply adds edges to each of the remaining vertices in every case.

Constructing $G(Z_n)$ for Z_n

Our first algorithm is parameterized by the order of a particular commutative ring Z_n and derives the zero-divisor graph $G(Z_n)$ of that ring. The algorithm is called *GenerateGraph*.

```
GenerateGraph(var G: graph,  n: integer)
        if n is prime
                return empty graph
        else if n=p^k
                GetGr(var G, n)
        else
                Factor n into two numbers p and q that are relatively prime to each
                other
                G1 = GenerateGraph(p)
                G2 = GenerateGraph(q)
                return ConstuctGraph(p,q,G1,G2)
```

The algorithm checks to see if the order of the ring, n, is a prime number. If so, the empty graph is returned, since there are no zero-divisors. If n is not prime, the algorithm checks to see if n is the power of some prime, p. If so, a call is made to another algorithm shown later to handle that case. Finally, if n is neither a prime nor a power of a prime, n is factored into two values, relatively prime to each other. There are algorithms for doing such factoring as standard in any first year programming course.

Once the factorization is complete, two recursive calls are made to *GenerateGraph* with the factors as arguments. The fundamental theorem of arithmetic guarantees us that the base case for this recursive algorithm will be reached and the algorithm will terminate. Finally, there is a call to ConstructGraph, which puts together the graphs obtained for p and q respectively.

First we will see how the zero divisor graph for a ring isomorphic to Z_n where $n = p^k$, and then we will look at the case when $n = pq$.

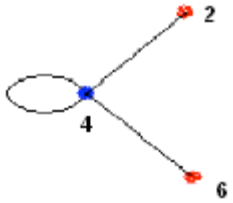
Here is the algorithm, *GetGr*, called when n is the power of a prime. GetGr constructs the zero divisor graph for Z_n where $n = p^k$.

GetGr (var G:graph, p^k : integer)
if k = 2
G = a connected graph(each node is connected to every other node including itself) with p-1 nodes labeled with the p-1 members of Z_p that are zero divisors.
else
GetGr(G, $p^{(k-1)}$)
Separate the nodes into sets according to the number of factors of p in each label, one set of elements whose factorization includes one factor of p, one set with a factor p^2,.., one set with elements whose factorization includes p^{k-2} .
Multiply every node label in each set by p.
Make the set of nodes with factor $\lceil k/2 \rceil$ form a connected subgraph.
Create one new set of nodes, those whose labels have only one factor of p and are zero divisors of Z_{p^k}
Connect each node from the set with k-j factors of p to every node from the set with j factors of p.

An example illustrates how this algorithm works:
To keep our graphs small, let's use $p = 2$ and follow the algorithm for $k = 2$, 3, and 4.
When $k = 2$, the algorithm returns the graph with $p - 1$ elements labeled with the appropriate value(s). For 2^2, there is only 1 zero divisor, namely 2, so the graph consists of only one node labeled 2.

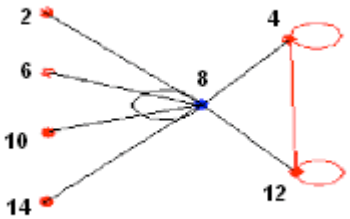
The graph of Z_4

When $k = 3$, the algorithm makes a recursive call to itself with parameter $k - 1$, i.e., 2 and gets the graph pictured above. Now the algorithm takes that graph and multiplies the label by p (2), getting a graph of one node labeled with 4. Then the algorithm divides the zero divisors for Z_8 into sets according to how many factors of p (2) each divisor has. In this case one set S1, has only one entry, S1={4}, while the other set S2 has two entries, S2={2, 6}. Next for values of j between 1 and 3, the algorithm connects nodes in the graph from any set whose elements have k-j factors of p to the elements from the set with j factors of p. In this case, the only value of j we need to consider is 2. S1 had entries with p^2 as a factor, while S2 has entries with p^1 as a factor. So, elements from S1 are connected to elements from S1, resulting in



The graph of Z_8

For the case when $k = 4$, the algorithm gets the graph just shown for $k = 3$, multiplies each label by 2, separates the nodes into sets S1 = {2, 6, 10, 14), S2 = {4, 12}, and S3 = {8}. Nodes from S1 are connected to those of S3, while the nodes in S2 are connected to each other.



The graph of Z_{16}

With the example as motivation, we are ready to discuss why this algorithm works. Given the order p^k of ring R, we can easily find the number of zero divisors and hence the number of nodes in the zero divisor-graph, namely $p^{(k-1)}$-1, using Euler's theorem. The labels on the nodes are the entries in Z_n that have a common divisor with n. If $k = 2$, then the zero-divisor graph is complete with p-1 nodes. Now if $k > 2$, then we can divide the nodes of $G(Z_n)$ into k-1 distinct sets of nodes, each set of nodes distinguished by the

factor of p its label has. There are k-1 different sets. Each node within a particular set has the same number of factors of p. In general, a set with whose labels have j factors of p has $I(j)=p^{k-j} - 1 - \sum_{h=j}^{k-1} I(k-h-1)$ nodes. How these nodes are connected to each other depends on the number of factors of p in the node labels. Every node that is in a set with a p factor number bigger or equal to $\lceil k/2 \rceil$ is connected to itself. Each node that is connected to itself is connected to every other node from the same set. This means that the nodes from each set that has nodes with number of p factors greater than $\lceil k/2 \rceil$ form a complete sub graph. A node with a particular number of p factors, x, is connected to a node with power y p-factors if $x + y \geq k$. This means if a node from a set with power x is connected to a to a node from a set with power y then every node from the set with power x is connected to every other node from the set with power y.

Now when we have a Z_n where $n=p^k$ we have p-1 elements that are divisible by p^{k-1} (they form a set with k-1 elements), p^2-1-(p-1) elements divisible by p^{k-2} (excluding those divisible by p^{k-1}) and they form the set of power k-2, p^3-1-(p^2-1+p-1) elements divisible by p^{k-3} (excluding those divisible by p^{k-1} and p^{k-2}) and they form the set of power k-3 and so on. If a node is of power j and $h<k$ such that $j+h>=k$ then the nodes from the set with power j are connected to the nodes from the set with power h(since p^k divides $p^j p^h$). If $j+j>=k$ then the nodes from the set with power j are connected among them-selves(since p^k divides $p^j p^j$). The GetGr algorithm is doing the following: it starts from the graph of p^2 and builds up gradually to p^k each time adding a new set of nodes with power one and drawing the proper edges to complete the zero-divisor graph for the current power of p.

For every two nodes x and y there are two possibilities: first x and y are from the same set, and second x and y are in different sets. If x and y are in the same set and they are connected the algorithm guaranties that x and y are in a set with power bigger or equal to ceiling(k/2). It also guaranties that all of these sets have their nodes connected to themselves. If x and y are from different sets then the algorithm guaranties that these nodes are connected if and only if their powers add up to a number bigger than or equal to k. From what we said above follows that the algorithm GetGr correctly derives the zero divisor graph of cyclic ring R of order p^k. Do we omit edges or nodes? No we do not omit nodes because all the numbers that are not relatively prime to $n=p^k$ are p^{k-1}-1, and we always have p^{k-1}-1 nodes. For a node x (divisible by p^h, where p^h is the biggest number divisible by p, x is divisible by) and a node y (divisible by p^j, where p^j is the biggest number divisible by p, y is divisible by) to be connected $p^j p^h$ should be divisible by p^k. In order for this to happen j+h should be bigger or equal to k. The algorithms guaranties that we are not going to omit an edge between any x and y. Keeping this is mind let's do an inductive proof.

Theorem1: The *GetGr* algorithm produces the zero-divisor graph for any Z_n ring, R, of order p^k where p is any prime and k is any integer bigger than 1.

Proof by induction:
Base Case: If we have a group of order p^2 then its zero divisor-graph is complete and has p-1 nodes.

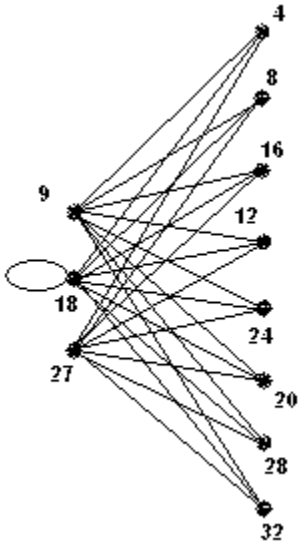
Inductive Step: Assume we have constructed the graph of Z_n, $n=p^{k-1}$. It has p^{k-2} -1 nodes, labeled with the zero divisors of $Z_{p^{k-1}}$. We want to construct the graph of Z_{p^k}. We first multiply the labels on the nodes by p, obtaining zero divisors in Z_{p^k}. Now add p^{k-1} –1-(p^{k-2} –1) additional nodes, each labeled with zero divisors of Z_{p^k} that have only one factor of p. Form k-1 sets, S, of labeled nodes, according to the number of factors of p in the label. Connect the nodes from set Si to set Sj if $i+j=k$. If the nodes of the set with power ceiling(k/2) do not form a clique make them form a clique. The newly formed graph has all of the zero-divisors of Z_{p^k} labeled correctly with edges connecting nodes whose labels yield zero when multiplied. By induction, the GetGr algorithm correctly constructs the zero divisor graph for rings of the form Z_{p^k}.

Now we turn to the third case, finding the zero divisor graph for Z_n when n is neither a prime nor a power of a prime. In this case, n is factored into two relatively prime factors, p and q. Z_n is represented by its isomorphic ring $Z_p \otimes Z_q$. Zero divisors are found for Z_p and Z_q. Finally, another algorithm, *ConstructGraph*, is called to construct the graph for Z_n.

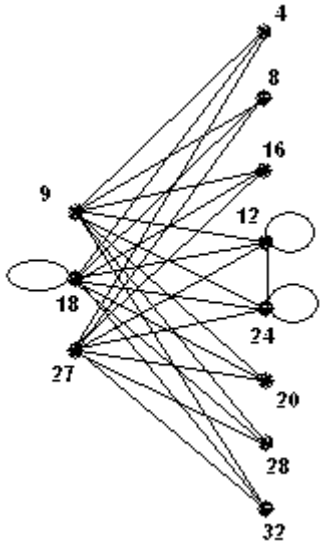
To motivate the algorithm, we look at an example. Suppose we want the graph for $Z_4 \otimes Z_9$. Calls to GenerateGraph provide a zero divisor graph of one node labeled with 2 for Z_4 and two nodes labeled 3 and 6 for Z_9. We will label our new graph for $Z_4 \otimes Z_9$ with pairs, of the form (x, y), x from Z_4 and y from Z_9.
We can use a two dimensional array to visualize the elements of this ring:

(0, 0) (1, 0) (2, 0) (3, 0)
(0, 1) (1, 1) (2, 1) (3, 1)
…
(0,8) (1, 8) (2, 8) (3, 8)

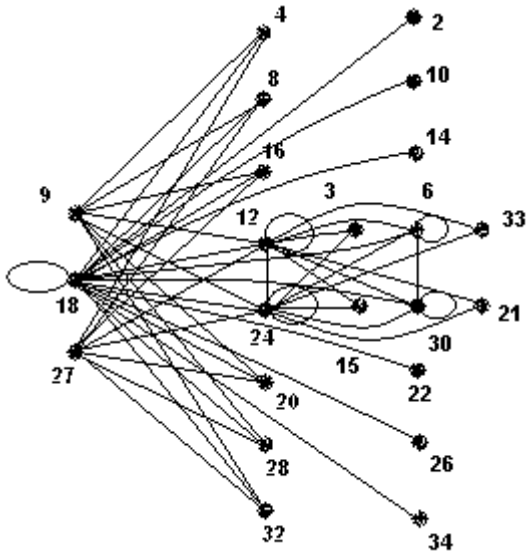
For constructing the graph, we omit the (0, 0) as a label. We connect nodes labeled with each element of the form (0, y) to every element of the form (x, 0). Then we form nodes labeled with pairs of the form (d, 0) where d is a zero divisor (a label in the graph for Z_4) and connect them with nodes labeled (e, 0) where $de = 0 \bmod Z_4$. To make the graph easier to read, rather than using the pairs as labels, we use the labels in Z_{36} corresponding to the pairs in $Z_4 \otimes Z_9$ under the canonical isomorphism between the two rings [4].



We also form nodes (0, a) and connect them to (0, b) where a□b = 0 mod Z_9.



Finally, we label nodes with pairs of the form (a, b) and (c, d) where a□b = 0 mod Z_4 and c□d = 0 mod Z_9.



Here is the algorithm:

ConstructGraph(var G : graph, G1: graph, G2: graph, p : integer, q : integer)

Get the zero divisors for Z_p and Z_q by using the labels from their graphs.

Create S1 = {(0,y) | y ∈ Z_q, y ≠0}.

Create S2 = {(x, 0) | x ∈ Z_p }.

To start graph G, connect every element of S1 to every element of S2.

For each zero divisor d, of Z_p, create a node labeled with the pair (d, 0) and connect (d, 0) to (e, 0) for every e such that d•e = 0 mod Z_p.

For each zero divisor a of Z_q, create a node labeled with the pair (0, a) and connect (0, a) to every (0, b) such that a•b = 0 mod Z_q.

Finally, add labels of the form (x, y) and connect to (w, z) where x•w = 0 mod Z_p and y•z = 0 mod Z_q.

Theorem 2 follows from the construction given in the algorithn:

Theorem 2. The ConstructGraph algorithm creates a graph for Z_p ⊗ Z_q where p and q are relatively prime.

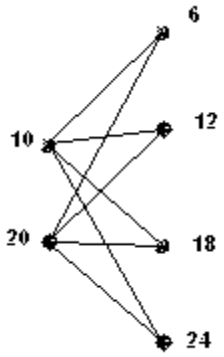
Constructing G(Z_n) for E_n

Finally, we define $\mathbf{E_n}$ to be the ring of even integers mod n, and we present an algorithm for find the zero divisor graphs for such rings. We consider cases for n as we did for Z_n.

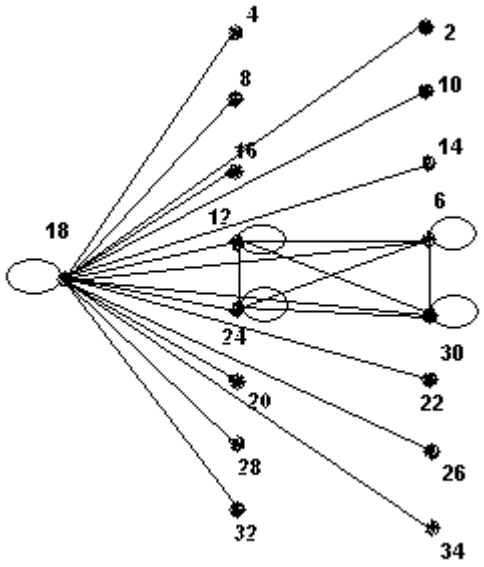
- Case 1. n is prime. As before, the zero divisor graph is empty.
- Case 2. $n = p^k$ for some k. This situation has two subcases, the case when p = 2 and the case when p is any odd prime. Case 1 takes care of p = 2. For higher powers of 2, the zero divisor graph is the same as that for Z_n. When p is an odd prime, the graph is empty, since no even integer can be a zero divisor for an odd integer.

- Case 3. n = pq where p and q are relatively prime. Without loss of generality, we can assume that all the powers of 2 are in the factor p. Then the zero divisor graph for Z_n can be obtained by getting the zero divisor graph for Z_p and then pruning out any edges that have one or more vertices labeled with an odd number.

To illustrate the algorithm, we include two sample graphs.



The graph of E_{30}



The graph of E_{36}

Summary

We have presented algorithms to create zero divisor graphs for rings isomorphic to Z_n or to E_n. The algorithms build the graphs of larger rings by using those of smaller rings. For those who are interested in addressing additional issues relative to these kinds of graphs, our algorithms have been implemented in C++ and a program for generating

graphs described here can be obtained from
http://www.denison.edu/~krone.

In future work we will consider zero divisor graphs for other kinds of rings such as polynomial rings, rings of matrices and other non-commutative rings.

References

1. Anderson, D. and Naseer, M., Beck's Coloring of a Commutative Ring, J. Algebra 159 (1993), 500-514.
2. Anderson, D. and Livingston, P., The Zero-Divisor Graph of a Commutative Ring, J.Algebra, 217 (1999), 434-447.
3. Beck, I., Coloring of Commutative Rings, J. Algebra 116 (1988), 208-226.
4. Herstein, I.N., Abstract Algebra, John Wiley & Sons, Inc., New York, 1999.
5. Kaplansky, I., "Commutative Rings," rev.ed., Univ. of Chaicago Press, Chicago, 1974.
6. Koh, K, "On Properties of rings with a finite number of zero-divisors," Math. Ann. 171 (1967), 79-80.