Hindawi
Journal of Applied Mathematics
Volume 2019, Article ID 4073905, 7 pages
https://doi.org/10.1155/2019/4073905

Research Article

Rainbow Connectivity Using a Rank Genetic Algorithm: Moore Cages with Girth Six

J. Cervantes-Ojeda, M. Gómez-Fuentes, D. González-Moreno, and M. Olsen

Universidad Autónoma Metropolitana, Cuajimalpa 05348, Mexico

Correspondence should be addressed to M. Gómez-Fuentes; mcgomezfuentes@aim.com

Received 12 September 2018; Accepted 29 January 2019; Published 3 March 2019

Academic Editor: Ali R. Ashrafi

A *rainbow t-coloring* of a t-connected graph G is an edge coloring such that for any two distinct vertices u and v of G there are at least t internally vertex-disjoint rainbow (u, v)-paths. In this work, we apply a Rank Genetic Algorithm to search for rainbow t-colorings of the family of Moore cages with girth six $(t; 6)$-cages. We found that an upper bound in the number of colors needed to produce a rainbow 4-coloring of a $(4; 6)$-cage is 7, improving the one currently known, which is 13. The computation of the minimum number of colors of a rainbow coloring is known to be NP-Hard and the Rank Genetic Algorithm showed good behavior finding rainbow t-colorings with a small number of colors.

1. Introduction and Definitions

Evolutionary algorithms have been applied to a wide variety of engineering problems [1], and they have also been applied to mathematics problems. For instance, Jong and Spears [2] showed that Genetic Algorithms (GA) can be used to solve NP-Complete problems, [3] applied a GA to a geometry problem, and [4] solved nonlinear algebraic equations by using a GA. Here we have successfully applied a Rank Genetic Algorithm (Rank GA) [5] to the graph theory problem of finding the rainbow connection number of a graph ($rc(G)$). Chakraborty et al. [6] proved that, for a given graph G, deciding whether $rc(G) = 2$ is NP-Complete and that it is also NP-Complete to decide whether a given edge-colored (with an unbounded number of colors) graph is rainbow connected. Therefore, our motivation is to try using the Rank GA heuristic on this problem. To our best knowledge, this problem has not been approached in this way nor using any other heuristic algorithms.

In a genetic algorithm we have an initial population of individuals; in our case, each individual represents a particular edge-colored graph. Each individual of the population is evaluated according to a fitness function; this function measures the ability of the individual for reaching a predetermined objective. Once individuals are evaluated, the next generation is obtained by applying genetic operators to the population inspired by the evolution in nature, such as crossover between individuals, selection of the fittest individuals, and mutations. This procedure is repeated until the genetic algorithm finds an individual that achieves the required objective or until a maximum number of generations are reached. A genetic algorithm can solve mathematical problems that are intractable by exhaustive search, as is the case of the problem described here.

A graph G is t-connected if and only if there are at least t internally disjoint (u, v)-paths connecting every two distinct vertices u and v of G [7]. A *rainbow path* is a path P such that all the edges of P have different colors (a path P of G is a sequence of distinct vertices $(v_1, v_2, \ldots, v_n)$ such that $\{v_i, v_{i+1}\}$ is an edge for $i \in \{1, 2, \ldots, n-1\}$). An edge coloring of a graph is called a *rainbow t-coloring* if for every pair of distinct vertices u and v there are at least t internally disjoint rainbow (u, v)-paths. The *rainbow t-connectivity* $rc_t(G)$ (defined by Chartrand et al. [8]) of a graph G is the minimum integer j such that there exists an edge coloring using j colors which is a rainbow t-coloring. The rainbow 1-connectivity is known as the *rainbow connection number* and was introduced by Chartrand et al. [9].

The rainbow t-connectivity of a graph has applications in the Cybernetic Security (see Ericksen [10]). For any fixed

$k \geq 2$, deciding if $rc_1(G) = k$ is NP-Complete [6, 11]. For more references on rainbow connectivity and rainbow t-connectivity we refer the reader to the survey by Li [12] et al. and the book of Li and Sun [13].

A graph G is *k-regular* if every vertex of G has degree k. The *girth* of G is the length of a shortest cycle of G. Given two integers $k \geq 2$ and $g \geq 3$ a $(k; g)$-*cage* is a k-regular graph with girth g and the minimum possible number of vertices. Let $n(k; g)$ denote the order of a $(k; g)$-cage. For references on cages see the dynamic survey of Exoo and Jajcay [14]. By counting the vertices emerging from a vertex or from an edge the lower bound $n_0(k; g)$, known as the *Moore bound*, is obtained. That is, $n(k; g) \geq n_0(k; g)$, where

$$n_0(k; g) = \begin{cases} 2\sum_{i=0}^{(g-2)/2} (k-1)^i = 2(k-1)^{g/2} - 2k - 2 & \text{if } g \text{ is even,} \\ 1 + \sum_{i=1}^{(g-1)/2} k(k-1)^{i-1} = \dfrac{k(k-1)^{(g-1)/2} - 2}{k-2} & \text{if } g \text{ is odd.} \end{cases} \quad (1)$$

When $n(k; g) = n_0(k; g)$ the $(k; g)$-cage is called a *Moore* $(k; g)$-*cage*; for references on Moore cages see [15]. The Moore cages have been characterized [16–18]. The existence of $(k; 6)$-Moore cages is related to the existence of finite geometries. For instance, the $(3; 6)$-cage is the incidence graph of Fano's plane. Concerning connectivity of $(k; 6)$-cages it has been proved that they are k-connected [19].

Chartrand et al. [20] showed that the rainbow 3-connectivity of the $(3; 6)$-cage (the Moore cage known as Heawood graph) is between 5 and 7 inclusive. Recently, Balbuena et al. [21] proved that it is not 5, and they bounded the rainbow k-connectivity of $(k; 6)$-cages as follows: if G is a Moore $(k; 6)$-cage, then

$$k \leq rc_k(G) \leq k^2 - k + 1 \quad (2)$$

If G is a $(k, 6)$-cage, then for $k = 3$, $k = 4$, and $k = 5$, respectively, we have $rc_3(G) \leq 7$, $rc_4(G) \leq 13$, and $rc_5(G) \leq 21$, respectively. In this paper we use a genetic algorithm to search for rainbow k-colorings of $(k; 6)$-cages with $k = 3$ and $k = 4$ [21] in order to see if the upper bound for $rc_k(G)$ can be improved.

The structure of this paper is as follows. In Section 2 we explain how the Rank GA was applied to the rainbow coloring problem. Results are presented in Section 3 and finally the conclusions are drawn in Section 4.

2. Finding Rainbow Colorings in a Moore Cage with a Genetic Algorithm

Finding a particular rainbow t-coloring of a graph G using r colors (that is a coloring with t internally disjoint rainbow paths between any pair of vertices) implies that $rc_t(G) \leq r$. We use an adapted version of the Rank GA [5] to find rainbow colorings in a Moore cage. To do so, each individual in the population contains a list of the assigned colors of each edge in the graph and is initialized randomly. In the Rank GA, the individuals of the population are ranked from best to worst in terms of their fitness before the genetic operators (selection, recombination and mutation) are applied. The application of these genetic operators depends on the rank of each individual in the population. The top ranked individuals tend to vary less than the bottom ranked ones. This is to make the latter try to escape from local optima of the fitness function. Also, top ranked individuals tend to be cloned more than others who tend to disappear.

Table 1: Parameters of the Rank GA.

Description	Value
Population size	$= \lvert E(G)\rvert$
Number of Generations	No limit
Population's selective pressure S	3

The adapted RankGA pseudocode that was used is given in Algorithm 1 and its parameters are shown in Table 1.

2.1. The Fitness Function. We detail below the fitness function. As this is an interdisciplinary work, we express the fitness function mathematically (below) and also in a computational way in Algorithm 2.

Let G be a graph, let $\Psi(G)$ be the set of all possible colorings of the edges $E(G)$, and let A be the set of all pairs of vertices of G, that is, $A = \{\{a_1, a_2\} \mid a_1, a_2 \in V(G)\}$. The fitness function to be maximized using the RankGA is given as

$$coloringFitness : \Psi(G) \longrightarrow \mathbb{R}[0, 1] \quad (3)$$

$$coloringFitness(\psi) = \left[\prod_{\mathbf{a}\in\mathbf{A}} pairFitness(\mathbf{a}, \psi)\right]^{1/|A|} \quad (4)$$

where $\psi \in \Psi(G)$ and

$$pairFitness : A \times \Psi(G) \longrightarrow \mathbb{R}[0, 1] \quad (5)$$

$$pairFitness(\mathbf{a}, \psi) = \max_{\mathscr{P}^k_{\mathbf{a}} \in DPS^k(\mathbf{a})} \left\{dpsFitness\left(\mathscr{P}^k_{\mathbf{a}}, \psi\right)\right\} \quad (6)$$

where $DPS^k(\mathbf{a})$ is the set of all sets $\mathscr{P}^k_{\mathbf{a}}$ of k internally disjoint paths between the vertices in $\mathbf{a}$ and

$$dpsFitness : DPS^k(\mathbf{a}) \times \Psi(G) \longrightarrow \mathbb{R}[0, 1] \quad (7)$$

$$dpsFitness\left(\mathscr{P}^k_{\mathbf{a}}, \psi\right) = \left[\prod_{p\in\mathscr{P}^k_{\mathbf{a}}} pathFitness(p, \psi)\right]^{1/|\mathscr{P}^k_{\mathbf{a}}|} \quad (8)$$

where

$$pathFitness : \mathscr{P}^k_{\mathbf{a}} \times \Psi(G) \longrightarrow \mathbb{R}[0, 1] \quad (9)$$

$$pathFitness(p, \psi) = \frac{pathNumColors(p, \psi)}{|E(p)|} \quad (10)$$

```
procedure RankGA
    numIndividuals ⟵ |E(G)|
    PopulationInitialization
    Evaluation and Sort
    while not end Criteria met do
        RankSelection
        RankRecombination
        Evaluation and Sort
        RankMutation
        Evaluation and Sort
procedure Rank Selection
    clones ⟵ null
    for i in [0..numIndividuals − 1] do
        r ⟵ i/(numIndividuals − 1)
        n ⟵ ⌊S(1 − r)^(S−1)⌋
        for j in [0..n − 1] do
            clones.add(ind_i)                    ▷ ind_i is cloned n times
    i ⟵ 0
    while clones.size() < numIndividuals do
        r ⟵ i/(numIndividuals − 1)
        n ⟵ S(1 − r)^(S−1)
        f ⟵ n − ⌊n⌋                              ▷ f is the fractional part of n
        if random(0, 1) < f then
            clones.add(ind_i)                    ▷ one extra clone of ind_i
        i ⟵ (i + 1) mod numIndividuals
    ind ⟵ clones                                 ▷ replace population
    Sort
procedure Rank Recombination
    for i in [0..numIndividuals − 2] step 2 do
        for j in [0..numGenes − 1] do
            if random(0, 1) < 0.5 then
                Switch(g_{i,j}, g_{i+1,j})       ▷ mating is with ind_{i+1}
procedure Rank Mutation
    for i in [0..numIndividuals − 1] do
        r ⟵ i/(numIndividuals − 1)
        for j in [0..numGenes − 1] do
            if random(0, 1) < r then
                g_{i,j} ⟵ random integer in the range [0..numColors − 1]
```

Algorithm 1: Rank GA.

where $E(p)$ is the set of all edges in path p and

$$pathNumColors : \mathscr{P}_{\mathbf{a}}^{k} \times \Psi(G) \longrightarrow \mathbb{R}[0,1] \tag{11}$$

$$pathNumColors(p,\psi) = \left| \bigcup_{e \in E(p)} \{\psi(e)\} \right| \tag{12}$$

where $\psi(e)$ is the color that the coloring ψ assigns to edge e.

The function *pathNumColors*, in (12), calculates the cardinality of a set made out of the colors $\psi(e)$ assigned to the edges $e \in E(p)$ in path p. The function *pathFitness*, in (10), calculates the ratio of the *pathNumColors* over the length of path p, being this a value in the range $[1/|E(p)|, 1]$, yielding 1 only in case the path has a rainbow coloring and lower values as the number of colors used in the path is reduced. The function *dpsFitness*, in (8), calculates the geometric mean of all the *pathFitness* values of all the $k + 1$ paths p in the disjoint path set $\mathscr{P}_{\mathbf{a}}^{k}$, yielding a value in the range $[0, 1]$ with 1 representing a rainbow disjoint path set. The function *pairFitness*, in (6), returns the maximum *dpsFitness* that can be obtained from the set of disjoint path sets $DPS^{k}(\mathbf{a})$ between the vertices in the vertex pair $\mathbf{a}$ yielding 1 if and only if there exists *at least* one rainbow disjoint path set $\mathscr{P}_{\mathbf{a}}^{k}$ in $DPS^{k}(\mathbf{a})$. Finally, the function *coloringFitness* in (4) computes the geometric mean of all the *pairFitness* values yielding a result that is 1 if and only if all pairs of vertices $\mathbf{a}$ have a fitness value of 1; i.e., the given coloring ψ is a rainbow coloring.

The pseudocode of the fitness function is given in Algorithm 2.

2.2. Finding the Set of Internally Disjoint Paths Sets between Pairs of Vertices in a Graph. The total number of possible sets $P_{\mathbf{a}}^{k}$ of k paths (internally disjoint or not) between the vertices in pair $\mathbf{a}$ is very high as k grows so in order to save computing time when calculating the fitness of an individual,

```
function coloringFitness(coloring)
    result ⟵ 1
    for Each vertex pair a in the Graph do
        r ⟵ pairFitness(a, coloring)
        result ⟵ result ∗ r                          ▷ Accumulate the product
    return result
function pairFitness(a, coloring)
    result ⟵ 0
    for Each InternallyDisjointPathsSet DPS^k of pair a do
        r ⟵ dpsFitness(DPS^k, coloring)
        if r > result then
            result ⟵ r                               ▷ Select best fitness
    return result
function dpsFitness(DPS^k, coloring)
    result ⟵ 1
    for Each Path p in DPS^k do
        r ⟵ pathFitness(p, coloring)
        result ⟵ result ∗ r                          ▷ Accumulate the product
    return result
function pathFitness(p, coloring)
    result ⟵ pathNumColors(p, coloring)/path.size
    return result
function pathNumColors(p, coloring)
    count ⟵ 0
    for Each Step s in p do
        c ⟵ coloring[s.source][s.dest]
        if not used[c] then                          ▷ count colors only once
            count ⟵ count + 1
        used[c] ⟵ true                               ▷ mark color as already used
    return count
```

Algorithm 2: Coloring fitness function.

the function *pairFitness* in (6) iterates only on the elements of the set $DPS^k(\mathbf{a})$ which is the set of path sets $\mathscr{P}^k_{\mathbf{a}}$ that contain k internally disjoint paths between the vertices in **a**. Algorithm 3 describes a way to calculate and store the set of all $DPS^k(\mathbf{a})$.

In this algorithm, the procedure *storeDisjointPathsSets* takes each vertex pair **a** in G and calls function *getPaths* in order to get the full set $P_{\mathbf{a}}$ of paths p between the vertices in **a** with length $l \leq numColors$ (since longer paths cannot be rainbow colored). Then it calls another function *getDisjointPathsSets* which tests each set $P^k_{\mathbf{a}}$ containing k paths $p \in P_{\mathbf{a}}$ between the vertices in **a** to see if $P^k_{\mathbf{a}}$ is a disjoint paths set and returns the set $DPS^k(\mathbf{a})$ of disjoint paths sets $\mathscr{P}^k_{\mathbf{a}}$.

The function *getPaths* is recursive and takes 3 parameters: a vertex pair $\mathbf{a} = \{a_1, a_2\}$, a graph G, and a maximum length *maxLength* of the paths to be found. Figure 1 illustrates how this function works. Paths that go from vertex a_1 to vertex a_2 are made by adding the step from a_1 to $neighbor_i$ of a_1 to all paths that go from $neighbor_i$ of a_1 to a_2 for each i. But before doing this, vertex a_1 is marked as visited in order to avoid paths that visit a vertex more than once. After computing the set of paths, vertex a_1 is unmarked.

In Table 2 we show how several measures grow with parameter k. First there is the number of vertices $|V(G)|$ in the $(k; 6)$ cage followed by the total number of vertex pairs. Then there is the number of vertex pairs with distances $d = 1$, $d = 2$, and $d = 3$ between them, respectively. We can see how the total number of paths with length $l \leq 7 = numColors$ between vertex pairs changes with distance d and with k. The next measure is the important one because it shows the drastic increase with k of the total number of paths sets containing k paths (internally disjoint and noninternally disjoint) that can be formed with the paths between the pairs of vertices with distances $d = 1$, $d = 2$, and $d = 3$ between them, respectively. All these sets need to be tested to see which of them are disjoint pats sets. Finally we show the total number of paths sets that need to be tested and the approximate time to compute this with our existing algorithms and computing power. As one can see the numbers are much bigger as k increases making it practically impossible, at the moment, to do the work for $k = 5$.

3. Results

The genetic algorithm was able to find several rainbow colorings of the $(k; 6)$-Moore cage for $k = 3$ and $k = 4$ using 7 colors. Using 6 colors, the algorithm could not find any rainbow colorations for neither $k = 3$ nor $k = 4$. A sample rainbow coloring that was found can be seen in Figure 2.

```
procedure storeDisjointPathsSets(G)
    initVertexNeighbors( )                          ▷ set the neighbors of each vertex for current k
    for Each a ∈ A = {{a1, a2}|a1, a2 ∈ V(G)} do
        paths[a] ⟵ getPaths(a, G, numColors)
        disjointPathsSets[a] ⟵ getDisjointPathsSets(paths[a], k)
function RECURSIVEgetPaths(a, G, maxLength)
    if maxLength ≤ 0 then                           ▷ wrong length
        return null
    if visited[a1] then                             ▷ a1 has already been visited
        return null                                 ▷ No paths to a2
    if a1 = a2 then                                 ▷ destination reached
        p ⟵ {a2}                                    ▷ single vertex in path
        return {p}                                  ▷ single path in the set
    visited[a1] ⟵ true
    for i = 1 to k + 1 do
        nextPair ⟵ {neighbor[i](a1), a2}            ▷ i^th neighbor of a1
        subResult ⟵ getPaths(nextPair, G, maxLength − 1)
        for Each path p in subResult do
            newPath ⟵ {a1}⌢p                        ▷ concatenate paths
            result ⟵ result ∪ newPath               ▷ aggregate path
    visited[a1] ⟵ false
    return result
function getDisjointPathsSets(Pa, k)
    for Each path set P_a^k of k paths p ∈ Pa do     ▷ may have huge # of iterations
        visited ⟵ false
        for Each internal vertex v of p do
            if visited[v] then
                next P_a^k
            visited[v] ⟵ true
        result ⟵ result ∪ P_a^k
    return result
```

Algorithm 3: Disjoint paths sets between vertices.

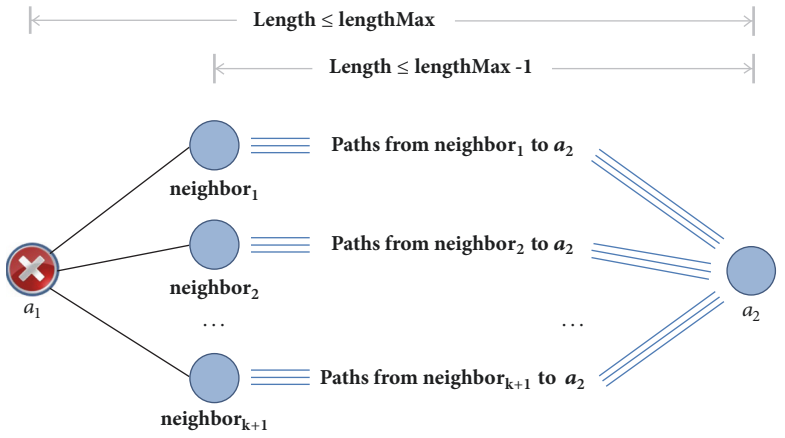

Figure 1: Recursive finding of the set of paths between vertices a_1 and a_2.

4. Conclusions

The problem of finding the rainbow k-connectivity $rc_t(G)$ of $(k;6)$-cages is intractable by exhaustive search. So it had not been possible to verify if it is possible to improve the upper bound given by (2). With the Rank GA we could see that it is unlikely that the upper bound for $(3;6)$-cages is less than 7, since with 6 colors no solution was found after having let the algorithm run for a long time. We also found that for the $(4;6)$-cage there is an upper bound of 7 colors since the algorithm found rainbow colorings in this case. This upper bound improves quite a lot the 13 colors given by (2) which binds the rainbow k-connectivity of $(k;6)$-cages with a quadratic function. By finding out that the upper bound for the $(4;6)$-cage is 7 we know that this upper bound function could be one that grows slower than a quadratic

Table 2: Growth with k of the effort to find the sets of disjoint paths sets.

k	3	4	5
$\|V(G)\|$	14	26	42
vertex pairs	91	325	861
vertex pairs with $d = 1$	21	52	105
vertex pairs with $d = 2$	42	156	420
vertex pairs with $d = 3$	28	117	336
paths for pairs with $d = 1$	17	136	642
paths for pairs with $d = 2$	17	82	257
paths for pairs with $d = 3$	33	244	1,025
paths sets for pairs with $d = 1$	680	13,633,830	887,805,286,368
paths sets for pairs with $d = 2$	680	1,749,060	8,984,340,696
paths sets for pairs with $d = 3$	5,456	144,084,501	9,336,731,022,080
total effort	195,608	17,839,699,137	3,234,134,601,579,840
approx. time	32 ms	23.1 min	8 years

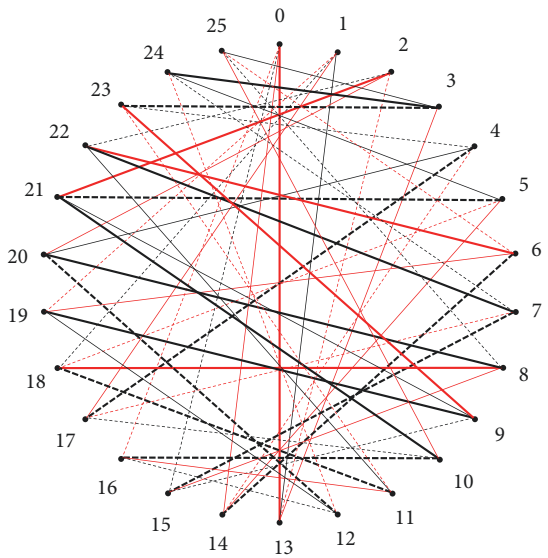

Figure 2: Sample rainbow coloring with $k = 4$ and $numColors = 7$.

function and may be much slower. To find the form of this function the problem should be scaled to Moore cages with $k \geq 5$; however, for $k = 5$, the problem is computationally very expensive, with the algorithms and computing power that we currently use. Thus, as for future work we will consider different approaches such as taking into account the symmetries of the graph, parallelization, and improvement of algorithms.

We believe that graph theory can benefit from the use of artificial intelligence techniques in some known NP-Complete and NP-Hard problems such as the one presented in this paper and there may be many of these cases.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they do not have any conflicts of interest.

Acknowledgments

This work was funded by Universidad Autónoma Metropolitana-Cuajimalpa. The authors would like to thank Dr. Rodolfo René Suárez Molnar, UAM-Cuajimalpa Unit Rector, for institutional support.

References

[1] D. Dasgupta and Z. Michalewicz, *Evolutionary Algorithms in Engineering Applications*, Springer Science and Business Media, 2013.

[2] K. A. De Jong and W. M. Spears, "Using genetic algorithms to solve NP-complete problems," *International Computer Games Association*, pp. 124–132, 1989.

[3] S. Jakobs, "On genetic algorithms for the packing of polygons," *European Journal of Operational Research*, vol. 88, no. 1, pp. 165–181, 1996.

[4] A. Pourrajabian, R. Ebrahimi, M. Mirzaei, and M. Shams, "Applying genetic algorithms for solving nonlinear algebraic equations," *Applied Mathematics and Computation*, vol. 219, no. 24, pp. 11483–11494, 2013.

[5] J. Cervantes and C. R. Stephens, "Limitations of existing mutation rate heuristics and how a rank GA overcomes them," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 369–397, 2009.

[6] S. Chakraborty, E. Fischer, A. Matsliah, and R. Yuster, "Hardness and algorithms for rainbow connectivity," in *Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science STACS*, pp. 243–254, 2009, Also, see Journal of Combinatorial Optimization, vol. 21, pp. 330–347, 2011.

[7] K. Menger, "Zur allgemeinen Kurventheorie," *Fundamenta Mathematicae*, vol. 10, pp. 96–115, 1927.

[8] G. Chartrand, G. L. Johns, K. A. McKeon, and P. Zhang, "The rainbow connectivity of a graph," *Networks. An International Journal*, vol. 54, no. 2, pp. 75–81, 2009.

[9] G. Chartrand, G. L. Johns, K. A. McKeon, and P. Zhang, "Rainbow connection in graphs," *Mathematica Bohemica*, vol. 133, no. 1, pp. 85–98, 2008.

[10] A. Ericksen, "A matter of security," *Graduating Engineer and Computer Careers*, pp. 24–28, 2007.

[11] V. B. Le and Z. Tuza, "Finding optimal rainbow connection is hard," 2009.

[12] X. Li, Y. Shi, and Y. Sun, "Rainbow connections of graphs: a survey," *Graphs and Combinatorics*, vol. 29, no. 1, pp. 1–38, 2013.

[13] X. Li and Y. Sun, *Rainbow Connections of Graphs*, Springer, London, UK, 2013.

[14] G. Exoo and R. Jajcay, "Dynamic cage survey," *The Electronic Journal of Combinatorics*, vol. DS16, 2013.

[15] M. Miller and J. Siran, "Moore graphs and beyond: a survey of the degree/diameter problem," *The Electronic Journal of Combinatorics - Dynamic Surveys*, vol. 14, 2005.

[16] E. Bannai and T. Ito, "On finite Moore graphs," *Journal of the Faculty of Science. University of Tokyo*, vol. 20, pp. 191–208, 1973.

[17] R. M. Damerell, "On Moore graphs," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 74, no. 2, pp. 227–236, 1973.

[18] A. J. Hoffman and R. R. Singleton, "On Moore graphs with diameters 2 and 3," *International Business Machines Journal of Research and Development*, vol. 4, pp. 497–504, 1960.

[19] X. Marcote, C. Balbuena, and I. Pelayo, "On the connectivity of cages with girth five, six and eight," *Discrete Mathematics*, vol. 307, no. 11-12, pp. 1441–1446, 2007.

[20] G. Chartrand, G. L. Johns, K. A. McKeon, and P. Zhang, "On the rainbow connectivity of cages," in *Proceedings of the Thirty-Eighth Southeastern International Conference on Combinatorics, Graph Theory and Computing*, vol. 184, pp. 209–222, 2007.

[21] C. Balbuena, J. Fresan, D. González-Moreno, and M. Olsen, "Rainbow connectivity of Moore cages of girth 6," *Discrete Applied Mathematics*, vol. 250, pp. 104–109, 2018.