

Lecture 2: Color Coding and Randomized Cycle Finding

Instructor: *Or Zamir*

Scribes: *Gal Wiener*

1 Introduction

Up to this point, we have discussed deterministic algorithms as well as those that rely on hash tables or hash sets, whose efficiency often depends on probabilistic assumptions. Today, however, we will explore probabilistic algorithms that explicitly incorporate randomness.

2 Probabilistic Pattern Detection

Claim 1. *If $m \geq n^{\frac{3}{2}}$, then a C_4 can be found in expected $O(m)$ time.*

Lemma 2. *Given an edge e , we can determine whether it is part of a C_4 in $O(m)$ time.*

Proof. Denote $e = (u, v)$:

Algorithm 3.

1. Identify the neighbors of u and v , the sets $N(u)$ and $N(v)$.
2. Find common neighbors $w_1, w_2 \in N(u) \cap N(v)$.
3. For each pair of common neighbors (w_1, w_2) , check if $(w_1, w_2) \in E$.
4. If such an edge exists, then $(u, w_1), (v, w_2), (w_1, w_2), (u, v)$ form a C_4 .

This check can be performed in linear time with respect to the number of edges, m .

□

Lemma 4. *In a graph that satisfies the condition in the claim ($m \geq n^{\frac{3}{2}}$), 99% of the edges are part of some C_4 .*

Proof. Let $G = (V, E)$ be a graph s.t. $|E| = m \geq n^{\frac{3}{2}} = |V|^{\frac{3}{2}}$

Algorithm 5.

1. Denote in E' the set of edges that are not part of any C_4 .
 - Therefore, $|E'| \leq n^{\frac{3}{2}}$.
2. Sample an edge e uniformly at random and check (in $O(m)$ time) whether it is part of a C_4 .
3. If it isn't, repeat steps 1-2 until we find one that is.

In expectation, we find such an edge in $O(1)$ steps.

□

3 Revisiting Pattern Detection

With the knowledge and techniques we discussed since being introduced to the problem of finding a C_4 in a graph, we can achieve tighter bounds for sparse(r) graphs:

Theorem 6. *Given a graph, it's possible to determine if it contains C_4 in $O(n^{\frac{4}{3}})$ time.*¹

Proof.

Algorithm 7. *We will perform operations on the graph $G = (V, E)$ and a copy of it $G' = (V', E')$.*

- If G' has a vertex v' with a $\deg(v') \geq m^{\frac{1}{3}}$, delete it.
 - For each e'_i incident to v' (and therefore deleted from G'), direct its equivalent edge in G , incident to its equivalent vertex $v - e_i$, **from** v outwards.
- Repeat until no such vertices are left in G' (Runs in linear time)

We split into two cases based on the algorithm's output:

- **Option 1:** The resulting G' after the algorithm finishes is *not* empty:

- It has, then, n' vertices, m' edges, and the the degree of any edge is at last $m'^{\frac{1}{3}}$.
- Note that

$$m' \geq \frac{1}{2} \cdot n' \cdot m'^{\frac{1}{3}} \cdot 100 \underset{m' \leq m}{\geq} \frac{100}{2} n' (m')^{\frac{1}{3}} \quad (1)$$

$$(m')^{\frac{2}{3}} \geq \frac{1}{2} n' \cdot 100 \quad m' \geq \left(\frac{100}{2} n'\right)^{\frac{3}{2}} \quad (2)$$

- In that case, we can find a C_4 in linear time. (✓)

- **Option 2:** All edges in G were *directed* such that the **out-degree** of every edge² is $\geq 100m^{\frac{1}{3}}$

We'll now use this result to determine if G contains a C_4 :

Algorithm 8.

1. Initialize an empty set, S , to contain ends of paths of length 2 (P_2) in G .
2. For each $(u, v) = e \in E(G)$:
 - For each edge $q = (x, u), (v, y) \in E(G)$ incident to the vertices u, v that e connects, we'll construct an (s, t) tuple of its ends. e.g. $(x, v), (u, y)$.
 - * If $(s, t) \notin S$, add it to S : $S = S \cup (s, t)$, Continue.
 - * Otherwise, $(s, t) \in S$, i.e. there's a collision in the set, then there are two distinct P_2 s between two vertices in G , i.e. a C_4 . Answer "Yes".

¹Note that if $m = n^{\frac{2}{3}}$, like in the problems from the previous lecture, the bound evaluates to n^2

²Also known as the *Degeneracy* of a graph, in this case the degeneracy of G is $\geq 100m^{\frac{1}{3}}$

Clearly, if we answered "Yes", then we found a C_4 , but what guarantees that a C_4 *would* be found? Note that no matter how the edges of a C_4 are directed, it can be deconstructed into two distinct P_2 -s such that each path has an edge that's directed *outwards* from the middle vertex.

Then every C_4 would be found.

Runtime: $m \cdot 2 \cdot 100m^{\frac{4}{3}} = O(m^{\frac{4}{3}})$

□

4 New Goal: Find General P_k, C_k In A Graph

- We will see algorithms that run in T time but with probability of success p (which may be small, but their success, or lack thereof, can be validated).
- Note that if we perform $\frac{t}{p}$ iterations, then the chances of failure are $(1-p)^{\frac{t}{p}} \leq e^{-\frac{t}{p}}$ and the runtime becomes $\frac{t}{p} \cdot T$.

Claim 9. *Given a graph G and a vertex v , we can determine if there exists a path P_k starting from v with high probability in $O(k! \cdot m)$ time.*

Proof.

Algorithm 10.

1. Randomly assign a linear ordering to the vertices of the graph G .
2. Ensure that the specified vertex v is the first vertex in this ordering.
3. Direct each edge (u, w) in the original graph G from u to w if u appears before w in the assigned ordering, or from w to u if w appears before u .
4. The resulting directed graph is acyclic (DAG)³

Observe that we can determine if there exists a simple path of length k in the DAG starting from v in $O(m)$:

Dynamic Programming Approach:

1. Initialize $D(v) = 0$ and $D(u) = -\infty$ for all other vertices $u \neq v$.
2. Process the vertices of the DAG in topological order.
3. For each vertex u , we compute $D(u)$, the length of the longest simple path ending at u that originates from v .
4. The value of $D(u)$ is given by:

$$D(u) = 1 + \max_{\{w | (w,u) \in E \text{ and } D(w) \neq -\infty\}} \{D(w)\}$$

where $D(w) = -\infty$ if there is no path from v to w .

³The constructed ordering is a topological sort.

If $\max_{u \in V} \{D(u)\} \geq k$, then a simple path of length k starting from v exists.

Runs in $O(m)$ time.

A specific path P_k from v is found if its vertices are ordered increasingly by the topological sort, which occurs with probability $\frac{1}{k!}$. Repeating the algorithm $100 \cdot k!$ times yields a failure probability of at most $(1 - \frac{1}{k!})^{100 \cdot k!} < e^{-100}$. \square

Observation 11. *Within the same time complexity, we could have checked for the existence of any path P_k originating from v , not just a specific one. To achieve this, we can augment the graph by adding a new vertex s and directed edges from s to every vertex in the original graph. Subsequently, searching for a P_{k+1} originating from s in this augmented graph is equivalent to searching for a P_k starting from v in the original graph.*

Question 12. *What if we wanted to determine, for every vertex $u \neq v$ in the graph, if there exists a path of length at least k from v to u ?*

Explanation: *If we repeat the algorithm $O(\log n \cdot k!)$ times, then the probability of failing to detect a path of length at least k from v to a specific vertex u is at most $\frac{1}{1000n}$. Consequently, the probability of making an error for at least one vertex $u \neq v$ (i.e., failing to detect a path that exists) is bounded above by $n \cdot \frac{1}{1000n} = \frac{1}{1000}$ using the union bound.*

4.1 Problem: Single-source, exact-length paths

Question 13. *Can we determine, for every vertex u , whether there exists a simple path of length exactly k from v to u ?*

Approach: *Utilizing the same dynamic programming approach. For each vertex u , we maintain a boolean array D_u of size $k + 1$, where $D_u[i]$ is true if and only if there exists a simple path of length i from v to u .*

4.2 Cycling back to cycles

Claim 14. *A C_k in a graph can be found in $O(k! \cdot k \cdot n \cdot m)$ time.*

Proof.

Algorithm 15.

1. For each vertex v in the graph, we perform the following:
 - (a) Run the randomized algorithm, with v as the source vertex.
 - (b) For each neighbor u of v in the original graph G , check if there is a path of length $k - 1$ from v to u in the directed acyclic graph constructed by the algorithm.
 - (c) If such a neighbor u is found, then the edge between u and v in G , combined with the path of length $k - 1$ from v to u , forms a cycle of length k .

Since the randomized algorithm from v takes $O(k! \cdot m)$ time and we iterate through at most n starting vertices v , and for each v we check its neighbors (at most n), the total time complexity is $O(n \cdot (k! \cdot m + n))$. \square

Claim 16. *We can find a P_k or a C_k in a graph within a time complexity of $O(k! \cdot \log k \cdot n^\omega)$.*

Proof. After assigning random vertex ordering (and therefore - random edge directions) the graph is guaranteed to be free of non-simple paths. Consequently, if we compute the $(k - 1)^{th}$ power of the adjacency matrix A of this DAG, the entry A_{ij}^{k-1} will be non-zero if and only if there exists a *simple path* of length $k - 1$ from vertex i to vertex j in the directed graph. This computation of A^{k-1} can be performed in $O(\log k \cdot n^\omega)$ time. \square

4.3 Improving the computational complexity factor of k

Remark 17. *The dependency of the previous claims and conclusions on $k!$ makes it impractical for large values of k . Therefore, a natural question arises: What is the greatest length of a simple cycle in a graph that we know how to compute in polynomial time -without a factorial dependence on k ?*

5 Color-Coding Method For Finding P_k, C_k

[Alon, Yuster, and Zwick (1995)]

The goal is improve upon the $k!$ factor in the time complexity, and aim for a dependence closer to $2^{O(k)}$ (i.e., the logarithm of the factor is linear in k). The Color-Coding method achieves this by replacing the random edge orientation technique with a random coloring of the vertices.

Instead of directing edges, we randomly assign one of k colors to each vertex in the graph. We will then demonstrate that it is computationally "easy" to detect "colorful" paths (or cycles), where each color appears at most once along the path (or cycle).

1. **Must be simple - like before.**
2. **What is the probability that a specific P_k in the graph is colorful?**

A P_k has k vertices. For it to be colorful, all k vertices must have distinct colors.

- There are $k!$ ways to assign k distinct colors to the k vertices of the path.
- There are k^k total ways to color the k vertices of the path using k colors (each vertex can get any of the k colors).

Therefore, the probability that a specific P_k is colorful is:

$$\frac{k!}{k^k}$$

Using Stirling's approximation ($k! \approx \sqrt{2\pi k} \cdot \left(\frac{k}{e}\right)^k$), we get:

$$\frac{k!}{k^k} \approx \frac{\sqrt{2\pi k} \cdot \left(\frac{k}{e}\right)^k}{k^k} = \frac{\sqrt{2\pi k}}{e^k}$$

Claim 18. *Given a graph with vertices colored using k colors and a starting vertex v , we can determine, for every vertex u , the set of lengths of all colorful paths starting at v and ending at u , with length at most k , in $O(2^{O(k)}mm)$ time.*

Proof. We proceed by dynamic programming on the length of the path. Let $C(i, u)$ be the set of color sets of colorful paths of length i that start at v and end at u .

For $i = 1$, for each neighbor u of v , $C(1, u) = \{\{c(v), c(u)\}\}$ if $c(v) \neq c(u)$, and $C(1, u) = \emptyset$ otherwise. For all other vertices w , $C(1, w) = \emptyset$.

For $i > 1$, we compute $C(i, u)$ as follows:

$$C(i, u) = \bigcup_{(w,u) \in E} \{S \cup \{c(u)\} \mid S \in C(i-1, w) \text{ and } c(u) \notin S\}$$

We iterate through path lengths i from 1 to k . In each iteration, we examine all edges in the graph. For each edge (w, u) , we iterate through the color sets in $C(i-1, w)$ and update $C(i, u)$ if adding the color of u maintains the colorful property.

Note: This is not a standard Breadth-First Search (BFS) because a vertex can be reached by colorful paths of the same length but with different sets of colors, requiring us to track these sets.

Finding C_k using Color-Coding: Similar to finding P_k , we can find a colorful C_k in $O(2^{O(k)}mn)$ time. The $2^{O(k)}$ factor stems from the fact that the DP state for a path of length up to k might involve tracking a subset of the k colors (for example, with a k -length boolean vector). \square

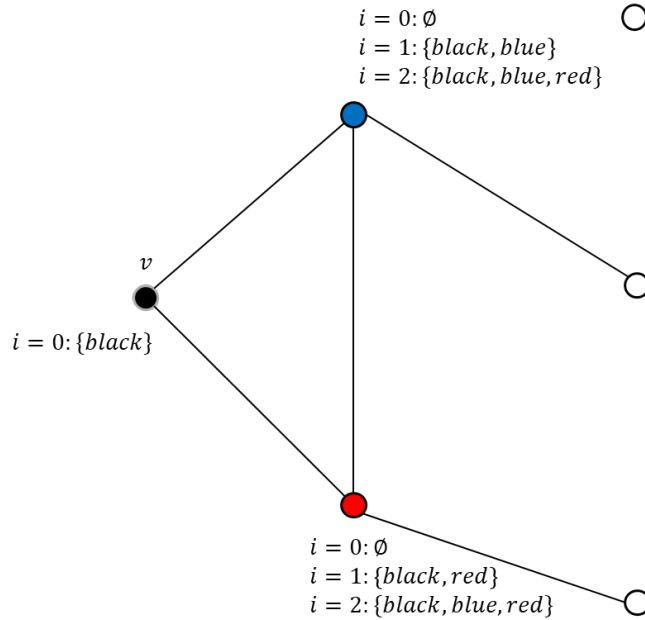


Figure 1: Example snapshot of the memory state of a Color-Coding procedure starting from v

Question 19. Can we achieve a runtime complexity of $O(2^{O(k)} \cdot n^\omega)$?

Algorithm 20. For each color $i \in \{1, \dots, k\}$, define a matrix A_i which is the adjacency matrix of the graph, modified as follows:

- Set $A_i[u, v] = 0$ if the color of v , $c(v) \neq i$.
- Set $A_i[u, v] = 0$ if the color of u , $c(u) = i$.

Also, define a matrix R_i such that for each color $i \in \{1, \dots, k\}$, let R_i be an $n \times n$ diagonal matrix defined as:

$$R_i[u, v] = \begin{cases} 1 & \text{if } u = v \text{ and } c(u) = i \\ 0 & \text{otherwise} \end{cases}$$

This definition ensures that when we multiply R_i by another matrix M , the resulting matrix $(R_i \cdot M)$ has rows corresponding to vertices of color i as they are in M , and all other rows are zero. That is, $(R_i \cdot M)[u, v] = M[u, v]$ if $c(u) = i$, and 0 otherwise.

Then, the number of paths from vertex i to vertex j that pass through colors $1, \dots, k$ in that specific order is given by:

$$\sum_{\sigma \in S_k} (R_{\sigma(1)} \cdot A_{\sigma(2)} \cdot A_{\sigma(3)} \cdots A_{\sigma(k)}) \quad (3)$$

where S_k is the set of all permutations of $\{1, \dots, k\}$.

Observation 21. *This approach is still problematic because it still has a $k!$ factor due to the summation over all $k!$ permutations $\sigma \in S_k$.*

We can also express this sum as:

$$\sum_{i=1}^k R_i \left(\sum_{\sigma \in S_{[k] \setminus \{i\}}} A_{\sigma(1)} \cdot A_{\sigma(2)} \cdots A_{\sigma(k-1)} \right) \quad (4)$$

The equality arises from considering all possible starting colors for the path. The left-hand side sums over all $k!$ permutations of the k colors. The right-hand side achieves the same by:

1. Choosing the first color i (using $\sum_{i=1}^k R_i$).
2. Summing over all possible orderings of the remaining $k - 1$ colors in the subsequent A matrix products (using $\sum_{\sigma \in S_{[k] \setminus \{i\}}} A_{\sigma(1)} \cdots A_{\sigma(k-1)}$).

Each term in the left-hand side corresponds to a unique combination of a starting color and an ordering of the remaining colors, which is accounted for in the right-hand side.

Claim 22. *Given a set of $k - 1$ matrices A_1, \dots, A_{k-1} , we can compute the sum:*

$$\sum_{\sigma \in S_{k-1}} \prod_{i=1}^{k-1} A_{\sigma(i)}$$

in $O(2^{k-1} \cdot n^\omega)$ time.

Proof. Dynamic programming: Let $D(S)$ be the sum of the products of matrices A_i for all permutations of the indices in the subset $S \subseteq [k - 1]$.

Base Case: For any subset S of size 1, say $S = \{i\}$, $D(S) = A_i$.

General Case: For a subset S with $|S| > 1$, we can express $D(S)$ by considering each element $j \in S$ as the first index in the permutation. Then, the remaining product is over the subset $S \setminus \{j\}$. Thus, we get:

$$D(S) = \sum_{j \in S} A_j \cdot D(S \setminus \{j\})$$

The number of subsets of $[k-1]$ is 2^{k-1} . For each subset, we iterate through at most $k-1$ elements to compute the sum, and each matrix multiplication takes $O(n^\omega)$ time. Therefore, the total time complexity is $O(2^{k-1} \cdot k \cdot n^\omega) = O(2^k \cdot n^\omega)$. \square

5.1 Results, so far:

- **Finding a cycle of length 3 (C_3):** Can be done in time $\min(n^\omega, m^{\frac{2\omega}{1+\omega}})$.
- **Finding a cycle of length 4 (C_4):** Can be done in time $\min(n^2, m^{4/3})$. Note that if $\omega = 2$ then $m^{\frac{2\omega}{1+\omega}} = m^{\frac{4}{3}}$.
- **Theorem:** For any integer $k \geq 2$, a cycle of length $2k$ (C_{2k}) can be found in a graph in $O(2^{O(k)} \cdot n^2)$ time.
- **Our latest result:** A cycle of length $2k+1$ (C_{2k+1}) can be found in $O(2^{O(k)} \cdot n^\omega)$ time.

6 Θ - Graphs

Definition 23. A Θ -graph (Clarkson, 1987)⁴ is a graph consisting of two cycles that share exactly one edge.

Definition 24. We say a Θ graph has girth greater than t if both of its constituent cycles have length strictly greater than t .

Theorem 25. If a graph G satisfies $m \geq 2t \cdot n$, then G contains a Theta graph with girth greater than t . Moreover, such a Θ graph can be found in $O(m)$ time.

Proof. We can find a subgraph of G with minimum degree at least $2t$ by iteratively deleting vertices with degree less than $2t$. This process takes $O(m)$ time.

Find a maximal simple path P in this subgraph. Let the endpoints of P be v_1 and v_k . This can be done in $O(m)$ time.

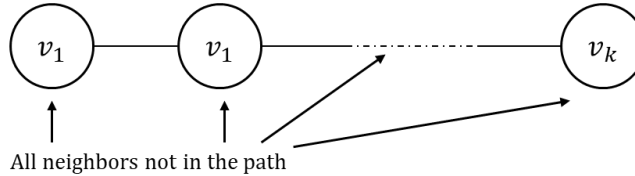


Figure 2: Construction of P

Consider the neighbors of v_1 , denoted by $N(v_1)$. Since P is a maximal simple path, all vertices in $N(v_1)$ must lie on the path P . Let the indices of these neighbors along the path be i_1, i_2, \dots, i_{2t} , where

⁴The K. Clarkson in question is not to be confused with Kelly Clarkson, born in 1982, who did not publish these conclusions when she was 5.

$1 \leq i_1 < i_2 < \dots < i_{2t} \leq k$. Since the path is maximal, and $m \geq 2t \cdot n$, the vertex v_1 has at least $2t$ neighbors on the path P . Let these neighbors be $v_{i_1}, v_{i_2}, \dots, v_{i_{2t}}$ in order of their indices on P .

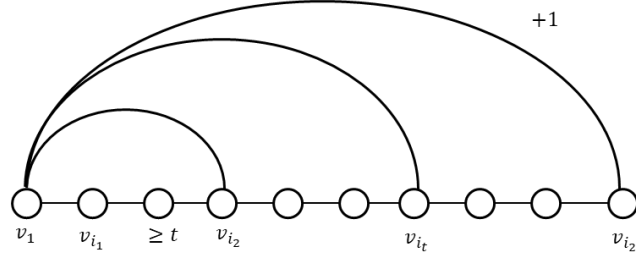


Figure 3: Construction of the maximal subgraph

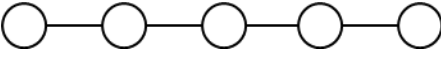
Consider the edges (v_1, v_{i_j}) for $j = 1, \dots, 2t$. These edges, along with the segments of the path P from v_1 to v_{i_j} , form cycles.

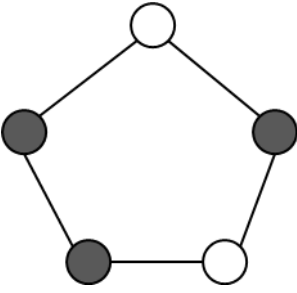
Consider the neighbors v_{i_t} and $v_{i_{2t}}$. The edges (v_1, v_{i_t}) and $(v_1, v_{i_{2t}})$, together with the path segment of P from v_1 to v_{i_t} and v_1 to $v_{i_{2t}}$ respectively, form two cycles.

The lengths of these cycles are at least t since the indices are i_t and i_{2t} and the path is simple and maximal. \square

Definition 26. Given a graph G and a coloring of its vertices, we say that the coloring is t -periodic if, for every path P_t in G , the two endpoints of the path have the same color.

Examples:

1.  Straight Line

2.  2-Periodic

Claim 27. Given a cycle C_l and a coloring of its vertices, the smallest t^* such that the coloring is t^* -periodic must divide l . (In other words, every cycle C_l is l -periodic.)

Proof. Let $t^* < l$ be the smallest period of the coloring. We can write l as $l = \lfloor \frac{l}{t^*} \rfloor \cdot t^* + (l \bmod t^*)$, where $0 \leq (l \bmod t^*) < t^*$.

If $l \bmod t^* = 0$, then we are done, as t^* divides l .

Assume for contradiction that $l \bmod t^* \geq 1$.

Consider two vertices in C_l that are a distance of $l \bmod t^*$ apart. We can also reach the second vertex from the first vertex by traversing the cycle in the other direction. This traversal consists of $\lfloor \frac{l}{t^*} \rfloor$ paths of length t^* . Since the coloring is t^* -periodic, all vertices along these paths have the same color as the starting vertex. Therefore, the two vertices at a distance of $l \bmod t^*$ apart must have the same color.

This implies that the coloring is also $(l \bmod t^*)$ -periodic. However, $l \bmod t^* < t^*$, which contradicts the assumption that t^* is the smallest period.

Therefore, $l \bmod t^* = 0$, and t^* divides l . □

Claim 28. *Consider a Θ -graph with girth greater than t , having a cycle of length l . Then, the smallest period t^* of a coloring for this Θ -graph is the same for all cycles in the graph.*

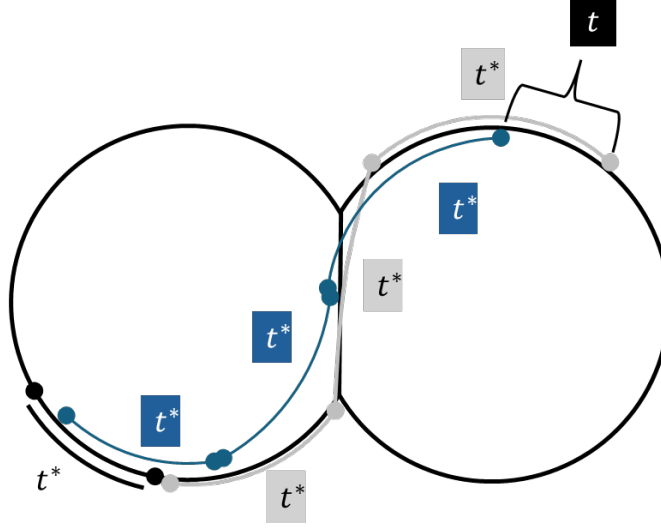


Figure 4: The entire graph is t -periodic

Proof. We can find paths of length $r \cdot t^*$ (for any integer r) that connect pairs of vertices at a distance of t^* on any other cycle of the Θ -graph. This would imply that the coloring has the same period for any cycle in the Θ -graph. □

Conclusion 29. *Suppose we have a Θ -graph with girth greater than t . Then its smallest period t^* is at least 2.*

Proof. We noted that any period $t^* \geq 1$ of one of the cycles is also a period of all cycles in the *entire* Θ -graph, and that the smallest period of any cycle must divide its length.

Let the lengths of the two cycles be $l_1 + 1$ and $l_2 + 1$, where the shared edge contributes 1 to each cycle. The length of the entire Θ -graph is $l_1 + l_2$.

Therefore,

$$t^* | l_1 + 1, t^* | l_2 + 1, \text{ and } t^* | l_1 + l_2 \quad (5)$$

$$\Rightarrow t^* | (l_1 + 1) + (l_2 + 1) - (l_1 + l_2) = 2. \quad (6)$$

Since the girth is greater than t , the lengths of the cycles are at least $t + 1$. Since the period divides the length of the cycle, and t^* is at least 1, if $t^* = 1$, then it will divide the cycle length, which is greater than t .

Therefore, t^* divides 2, and t^* must be at least 2. □

References

- [1] Noga Alon, Raphael Yuster, and Uri Zwick. “Color-coding”. In: *J. ACM* 42.4 (July 1995), pp. 844–856. ISSN: 0004-5411. DOI: 10.1145/210332.210337. URL: <https://doi.org/10.1145/210332.210337>.
- [2] K. Clarkson. “Approximation algorithms for shortest path motion planning”. In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. STOC '87. New York, New York, USA: Association for Computing Machinery, 1987, pp. 56–65. ISBN: 0897912217. DOI: 10.1145/28395.28402. URL: <https://doi.org/10.1145/28395.28402>.