Recognizing some corona products in polynomial time

Jarlilson Guajajara[1], **Julliano Rosa Nascimento**[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 131 – 74001-970 – Goiânia – GO – Brasil

jarlilsonguajajara@discente.ufg.br, jullianonascimento@ufg.br

Abstract. *The corona product of G and H is the graph $G \circ H$, obtained by taking a copy of G, $|V(G)|$ copies of H, and connecting the i-th vertex of G to each vertex in the i-th copy of H, where $1 \leq i \leq |V(G)|$. We present an algorithm for recognizing some corona product graphs, a structure that is particularly useful for analyzing hierarchical networks. Our results demonstrate that the algorithm works for arbitrary graphs G when H is restricted to classes where the isomorphism problem is solvable in polynomial time. Notably, for H as a path, cycle, or complete graph, the algorithm runs in cubic time.*

1. Introduction

Graph Theory is one of the cornerstones of Computer Science, providing tools for analyzing and solving complex problems. Its applications are widely used by information technology and data analysis professionals, as well as academic and corporate researchers, educators, and computer science students.

Two central aspects of Graph Theory are the categorization of graphs with common properties and the solving of problems using specific algorithms. There are tools that allow both direct interaction with graphs and the execution of algorithms on them. However, solutions such as those presented in [UnickSoft 2024] and [Halim 2015], while visually appealing, do not cover a wide range of fundamental algorithms for Graph Theory nor do they offer options for generating specific graph families.

To enhance graph identification and the application of algorithms for educational and scientific purposes, we plan to expand the open-source tool *web Graph Problems*, developed by [Silva 2018], by integrating functionality for identifying corona products in graphs. It is worth noting that previous improvements to this tool have already been made, as reported in [da Silva and da Silva 2022].

The corona product was introduced in 1970 by [Frucht and Harary 1970] as a operation to simplify the study of the relationship between graph structure and algebraic properties. Unlike the more complex lexicographic product [Sabidussi 1959], the corona product establishes a straightforward connection between the automorphism group of the resulting graph and the wreath product of the automorphism groups of the original graphs. This simplicity makes the corona product a practical and intuitive tool for analyzing composite graph structures and their symmetries.

There is a variety of studies addressing the corona product of graphs. For instance, [Sharma et al. 2019] investigates corona products as models for network-generating processes with hierarchical structure that incorporate property duplication. Additionally, some well-known problems in Graph Theory have been studied within

this product framework, such as domination in [G. Yero et al. 2012], equitable coloring in [Furmanczyk et al. 2012] and perfect coloring in [Bhange and Bhapkar 2020], among others.

According to [Feigenbaum and Schaffer 1985], deciding whether a given graph can be expressed as a corona product of two graphs is polynomially equivalent to the graph isomorphism test, even when G is connected. However, it is possible to verify whether a graph is a corona product within classes in which the isomorphism problem can be solved in polynomial time.

In this paper, we introduce an algorithm to recognize corona product graphs, allowing G to be an arbitrary graph and restricting H to the classes of paths, cycles, and complete graphs, achieving cubic complexity in these cases. Although the algorithm is currently in the theoretical stage, we plan to implement it in the future and integrate it into the open-source tool *web Graph Problems* [Graph problems tool 2022], expanding its functionality for analyzing and recognizing graph structures.

This paper is structured as follows. Section 2, we review the necessary theory to understand the developed algorithm; in Section 3, we describe the algorithm for identifying corona products, as well as its limitations; finally, in Section 4, we present our results.

2. Preliminaries

We will use the standard notations and definitions in Graph Theory as presented in [Bondy and Murty 2008] and [Booth and Colbourn 1979]. For a graph G, its vertex and edge sets are denoted by $V(G)$ and $E(G)$, respectively. Given two graphs G and G', we say that graph G is *isomorphic* to graph G' and write $G \cong G'$ if there exists a bijection $\phi : V \to V'$ such that $xy \in E(G)$ if and only if $\phi(x)\phi(y) \in E(G')$ for all $x, y \in V$.

In a simple graph of order n, the *degree* of a vertex v, denoted by $\deg(v)$, is the number of edges incident to v. An *isolated vertex* is a vertex of degree zero, that is, a vertex that is not an endpoint of any edge. The *neighborhood* of a vertex v, denoted by $N_G(v)$, is the set of all neighbors of v in G. The neighborhood of a subset V of $V(G)$, denoted as $N_G(V)$, is the union of the neighborhoods of the vertices of V. A *subgraph* of a graph G is a graph $G' = (V', E')$, where $V' \subseteq V(G)$ and $E' \subseteq E(G)$. If G' contains all edges $uv \in E(G)$ for all vertices $u, v \in V'$, then G' is called an *induced subgraph* of G, and is denoted by $G[V']$. We say that V' *induces* or generates G' in G.

A relation R on a set S is called *reflexive* if, for every $a \in S$, we have $a\,R\,a$, meaning that each element is related to itself. The relation R is *symmetric* if, for any $a, b \in S$, we have $a\,R\,b \Rightarrow b\,R\,a$; in other words, if an element a is related to b, then b is also related to a. The relation R is *transitive* if, for any $a, b, c \in S$, we have $a\,R\,b$ and $b\,R\,c \Rightarrow a\,R\,c$, which means that if a is related to b and b is related to c, then a is also related to c. If a relation R on S satisfies the properties of reflexivity, symmetry, and transitivity, then R is called an *equivalence relation* on S.

Two problems A and B are considered *polynomially reducible* if there is a polynomial-time computable function f that transforms any instance x of problem A into a corresponding instance $f(x)$ of problem B, such that $x \in A$ if and only if $f(x) \in B$. If there exists also a polynomial-time computable function in the reverse order, we say that

A and B are *polynomially equivalent*. In that case, solving either of the two problems in polynomial time implies that the other can also be solved in polynomial time.

We define a *loop* as a set of instructions that repeats within an algorithm, continuing for a specified number of iterations or until a certain condition is met to terminate the process.

2.1. Some Classes of Graphs

Classes of graphs group those that share common properties. In this work, we consider only undirected graphs. A *path* is a sequence of distinct vertices connected by edges between consecutive vertices. The first vertex in this sequence is called the *initial* vertex, while the last vertex is the *terminal* vertex. The shortest path between two vertices, known as the *shortest path*, contains the minimum number of edges connecting them.

A *path graph* is one formed by n vertices $v_1, v_2, \ldots, v_n$, where each vertex v_i is connected to the next vertex v_{i+1} by an edge, for $1 \leq i \leq n-1$. The vertices v_1 and v_n are the endpoints, each with degree one, while all other vertices have degree two.

On the other hand, when a path starts and ends at the same vertex, it forms a *cycle*. The *cycle graph*, denoted by C_n, has n vertices and n edges, with each vertex having degree two, provided $n \geq 3$.

The *complete graph*, by contrast, connects every pair of distinct vertices with an edge. Denoted by K_n, it contains $\frac{n(n-1)}{2}$ edges.

Finally, a graph G is said to be *bipartite* if its vertex set $V(G)$ can be divided into two disjoint subsets A and B, such that all edges connect vertices from A to vertices in B. The *complete bipartite graph*, $K_{n,m}$, is a bipartite graph where $|A| = n$ and $|B| = m$, and each vertex in A is connected to every vertex in B.

Let G be a graph with n vertices and H a graph with m vertices. The *corona product* of G and H is the graph $G \circ H$ constructed by taking a copy of G and n copies of H. The connection between them is established by adding an edge from the i-th vertex of G, for $1 \leq i \leq n$, to each vertex in the i-th copy of H. Throughout this paper, we refer to G as the *core* and H as the *copy*, these being the copies of H in $G \circ H$. See an example of a corona product in Figure 1.

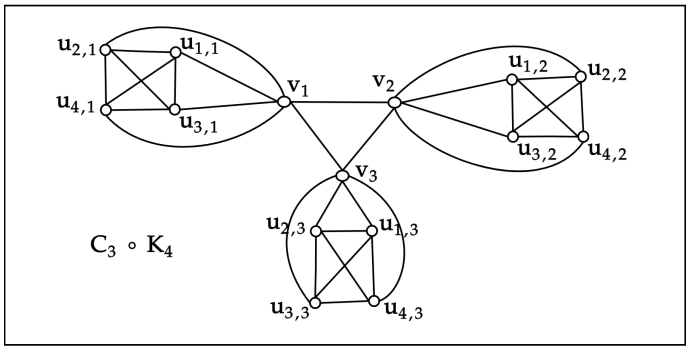

Figure 1. Example of the corona product of a cycle C_3 and a complete graph K_4, $C_3 \circ K_4$.

3. Recognition Algorithm

In this section, we present our main result, the Algorithm 1. It recognizes the corona product in graphs without restrictions on G, while H is restricted to graphs in classes where the isomorphism test can be resolved in polynomial time.

Algorithm 1: IsCoronaProduct(X)

Input: Connected graph $X = (V, E)$
Output: `True`, if there exists a corona decomposition $X \cong G \circ H$, for some graphs G and H; or `False`, otherwise

```
1  Let n = |V(X)|;
2  Let v_1, v_2, ..., v_n be an ordering of the vertices of X such that
     d(v_1) ≥ d(v_2) ≥ ··· ≥ d(v_n);
3  for i = 1 to n do
4      Let q = (n−i+1)/i;
5      if q ∈ ℤ then
6          Let V(G) = {v_1, v_2, ..., v_i};
7          for j = 1 to i do
8              if |N_X(v_j) \ V(G)| ≠ q then
9                  break this j-loop and begin the next iteration of the i-loop;
10             Let V(H_j) = N_X(v_j) \ V(G);
11         for j = 1 to i − 1 do
12             if X[V(H_j)] ≇ X[V(H_{j+1})] then
13                 break this j-loop and begin the next iteration of the i-loop;
14         return True;
15 return False;
```

First, Algorithm 1 attempts to identify the core of a potential corona product. For this purpose, we use Proposition 1. First, we recall the definition of the corona product. For each vertex $v_i \in V(G)$, a copy of the graph H, denoted by H_i, is added. Each vertex of H_i, denoted as $u_{j,i} \in V(H_i)$, where $u_{j,i}$ corresponds to $u_j \in V(H)$, is connected to the vertex v_i in G. It is also important to note that there are no connections between distinct copies of H.

Proposition 1. *Let $X \cong G \circ H$ be the corona product of two non-trivial connected graphs G and H. Denote $V(G) = \{v_1, \ldots, v_n\}$ and $V(H) = \{u_1, \ldots, u_m\}$. If $v_i \in V(G)$ and $u_j \in V(H)$, for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$, then $\deg_X(u_{j,i}) < \deg_X(v_i)$.*

Proof. We aim to prove that, for $v_i \in V(G)$ and $u_{j,i} \in V(H_i)$, the degree of $u_{j,i}$ in the graph $X = G \circ H$ is less than the degree of v_i.

To do this, let us analyze the degrees. The vertex $v_i \in V(G)$ is connected to at least one neighbor in G, as we define G as a non-trivial graph. If $\deg_G(v_i)$ is the degree of v_i in the graph G, then v_i has $\deg_G(v_i)$ neighbors. Additionally, v_i is connected to all vertices of the copy H_i, which contains m vertices, as $|V(H)| = m$. Hence, the degree of v_i in X is given by $\deg_X(v_i) = \deg_G(v_i) + m$. Now, we compare the degree of v_i with the

degree of $u_{j,i}$, where $\deg_X(u_{j,i}) = \deg_H(u_j) + 1$. Thus, for $\deg_{G\circ H}(u_{j,i}) < \deg_{G\circ H}(v_i)$ to hold, we must have $\deg_H(u_j) + 1 < \deg_G(v_i) + m$.

We know that $u_j \in V(H)$, so $\deg_H(u_j) \leq m - 1$, since the maximum degree of a vertex in H is $m-1$, in the case where H is a complete graph. This implies $\deg_H(u_j)+1 \leq m$.

On the other hand, $\deg_G(v_i) \geq 1$, as v_i must have at least one neighbor in G. Thus, $\deg_G(v_i) + m \geq m + 1$. Consequently, we conclude that $\deg_H(u_j) + 1 \leq m < \deg_G(v_i) + m$, which proves that $\deg_X(u_{j,i}) < \deg_X(v_i)$.

Therefore, we conclude that, for any vertex $u_{j,i} \in V(H_i)$ and $v_i \in V(G)$, the degree of $u_{j,i}$ in the graph $G \circ H$ is always less than the degree of v_i. □

By Proposition 1, we know that the vertices with greater degree in X are candidates to belong to the core. The following lemmas are derived from the Algorithm 1:

Lemma 1. *If $X \cong G \circ H$, then $|V(X)| = |V(G)| + |V(G)| \cdot |V(H)|$.*

Proof. This result follows directly from the definition of the corona product, which states that each vertex in G is associated with a distinct copy of H, forming a union of these copies alongside G itself. □

Thus, the structure permits the vertices to be arranged in a matrix representing the order and adjacency of vertices within the corona product.

Lemma 2. *The Isomorphism relation R between two graphs is an equivalence relation.*

Proof. To show that isomorphism is an equivalence relation, we must prove that this relation is reflexive, symmetric, and transitive.

Let G be a simple graph. The identity function $\mathrm{id} : V(G) \to V(G)$, defined by $\mathrm{id}(v) = v$ for every $v \in V(G)$, is a bijection from $V(G)$ to itself. Moreover, since id preserves adjacencies, i.e., $uv \in E(G) \Leftrightarrow \mathrm{id}(u)\,\mathrm{id}(v) \in E(G)$, we have that G is isomorphic to itself. Therefore, the isomorphism relation is reflexive.

Now, suppose G_1 and G_2 are simple graphs such that G_1 is isomorphic to G_2. Then, there exists a bijection $f : V(G_1) \to V(G_2)$ that preserves adjacencies, i.e., $uv \in E(G_1) \Leftrightarrow f(u)f(v) \in E(G_2)$. Since f is a bijection, there exists an inverse function $f^{-1} : V(G_2) \to V(G_1)$, which is also a bijection. Furthermore, f^{-1} preserves adjacencies, because if $f(u)f(v) \in E(G_2)$, then $uv \in E(G_1)$. Therefore, f^{-1} is an isomorphism from G_2 to G_1, which implies that G_2 is isomorphic to G_1. Thus, the isomorphism relation is symmetric.

Finally, let G_1, G_2, and G_3 be simple graphs such that G_1 is isomorphic to G_2 and G_2 is isomorphic to G_3. Then, there exist bijections $f : V(G_1) \to V(G_2)$ and $g : V(G_2) \to V(G_3)$ that preserve adjacencies. The composition $g * f : V(G_1) \to V(G_3)$ is a bijection, and for any $u, v \in V(G_1)$, we have $uv \in E(G_1) \Leftrightarrow f(u)f(v) \in E(G_2) \Leftrightarrow g(f(u))g(f(v)) \in E(G_3)$. Thus, $g * f$ preserves adjacencies, which implies that G_1 is isomorphic to G_3. Therefore, the isomorphism relation is transitive. □

If the conditions are satisfied, the algorithm performs final checks to ensure that the degree of the neighbors of the core vertices is compatible with the structure of a corona product. Otherwise, it concludes that X is not a corona product. If all conditions are met, the graph is recognized as such.

This set of heuristics explores both the global structure of the graph and local properties of vertices, allowing the algorithm to run efficiently, as we will prove in the following proposition.

Proposition 2. *Let X be a connected graph with n vertices. Algorithm 1 correctly determines whether X is a corona product $G \circ H$ or not in $O(n^3 + n^2 \cdot f(n))$ time, where $f(n)$ is the complexity of deciding isomorphism in the class of graph H.*

Proof. To perform an asymptotic analysis of Algorithm 1, we examine each execution step and its corresponding computational complexity to determine the total cost in terms of runtime.

Initially, the algorithm sorts the vertices $v_1, v_2, \ldots, v_n$ of the graph X in non-increasing order of degree, which requires a sorting operation with complexity $O(n \log n)$, where n is the number of vertices in X. This sorting serves as preprocessing for the following loops and will be done only once during the algorithm's execution. The ideia behind the sorting comes from Proposition 1.

Next, the algorithm enters a main loop in Line 3 that iterates over each possible value of i, where i ranges from 1 to n, exploring different vertex partitions to check if a corona product structure can be identified. Therefore, this loop executes n iterations in the worst case, as it goes through all possible choices for i.

Within the main loop, the algorithm computes the value $q = \frac{n-i+1}{i}$, an operation with constant complexity $O(1)$ that is performed in each iteration. This value q determines the feasibility of partitioning the vertices as intended, as non-integer values indicate an incompatible distribution for the base structure. When feasible, the resulting configuration can be visualized as a matrix, as shown in Figure 2, an arrangement justified by Lemma 1.

After defining q, the algorithm enters a second loop in Line 7, iterating over each vertex v_j in the set $V(G)$, which contains at most n vertices. At this point, for each vertex v_j, the algorithm checks if the neighborhood $N_X(v_j) \setminus V(G)$ contains exactly q vertices. This neighborhood check has a cost proportional to the degree of v_j, which is $O(n)$ in the worst case, as v_j could be connected to all other vertices in X. Thus, the cost of this operation for all n vertices in $V(G)$ is $O(n^2)$.

Finally, the algorithm performs isomorphism tests in Line 11 between the subgraphs H_j, verifying if they are isomorphic to each other. This test is conducted for $n-1$ pairs of subgraphs H_j, aiming to ensure that all copies H_j form the desired corona product structure. The isomorphism test between two graphs generally has a high cost, potentially requiring exponential time in the worst case. We denote this cost as $f(n)$, where $f(n)$ represents the complexity of deciding isomorphism. Given that there are $n-1$ pairs of graphs to compare—as we do not need to check all pairs since Lemma 2 guarantees that isomorphism is an equivalence relation—the complexity of this step is $O(n \cdot f(n))$.

Therefore, the total complexity of the algorithm is dominated by the main loop and the isomorphism tests, as these represent the most computationally expensive processes.

The overall complexity of the algorithm can be expressed as $O(n^3 + n^2 \cdot f(n))$, where $f(n)$ is the cost associated with the isomorphism test. If the subgraphs H_j are small or the isomorphism test can be optimized for $o(n^3)$, the term $O(n^3)$ becomes dominant. □

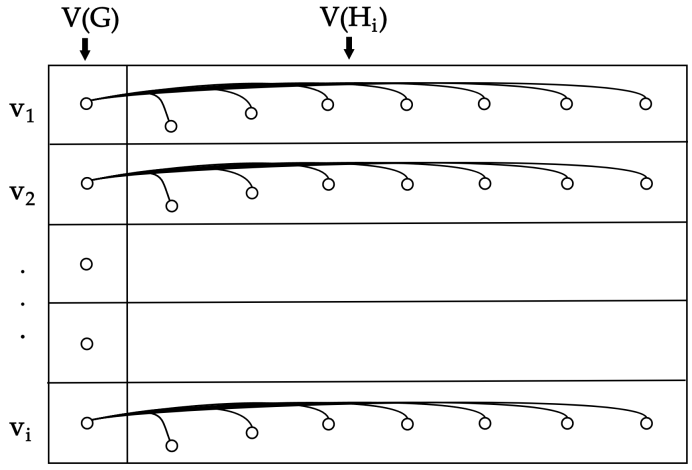

Figure 2. **Matrix $(q+1) \times i$ representing the partitioning process of Algorithm 1.**

Path graphs, cycles, and complete graphs exhibit regular structures with well-defined degree properties, which simplifies analysis in the context of the corona product. In these classes, vertex degrees follow specific patterns. In complete graphs, all vertices have a maximum and uniform degree. In path graphs, vertex degrees vary predictably between one and two. In cycles, all vertices have degree two. This degree regularity enables a simplified isomorphism verification among the copies of H and facilitates the decomposition into a corona product, due to the structural predictability of these classes. See how the corona product of a cycle C_3 and a complete graph K_4 is represented in the matrix in Figure 3.

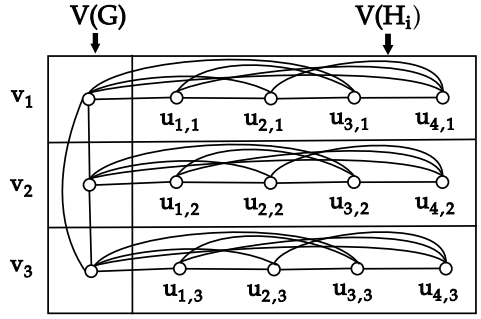

Figure 3. **For $q = 4$ and $i = 3$, the matrix $(q+1) \times i$, resulting in 5×3, illustrates a successful partitioning process of the corona product $C_3 \circ K_4$.**

From Propositions 1 and 2, we derive the following corollary, which formalizes the efficiency of the proposed algorithm for specific graph classes.

Corollary 1. *Let X be a graph that admits a corona decomposition $X \cong G \circ H$, where H belongs to the classes of path, cycle, or complete graphs. Under these conditions, corona product recognition can be performed with $O(n^3)$ complexity.*

Proof. Recall the result established in Proposition 2, which states that the recognition of the corona product $X \cong G \circ H$ can be performed with a complexity of $O(n^3 + n^2 \cdot f(n))$, where $f(n)$ depends on the complexity of verifying structural constraints for H.

For H being a path, a cycle, or a complete graph, $f(n) = O(n)$, which implies $n^2 \cdot f(n) = O(n^3)$. Thus, the total complexity reduces to $O(n^3)$.

This confirms that for these specific classes of H, corona product recognition is achievable within cubic time, as stated. □

This result demonstrates the algorithm's practicality when H belongs to well-structured graph classes, such as paths, cycles, or complete graphs. These graph classes enable the algorithm to leverage their inherent structural properties, ensuring efficient recognition within cubic time complexity. Future work could investigate whether similar performance guarantees can be extended to other graph classes.

4. Final Remarks

In this work, we presented an algorithm to recognize some corona product graphs. The approach allows G to be any arbitrary graph while restricting H to graph classes where the isomorphism problem is solvable in polynomial time.

This study represents an initial contribution to the recognition of corona products in graphs, providing an efficient solution for specific classes as seen in Corollary 1.

For future work, we suggest the implementation and integration of this algorithm into the system described in [Graph problems tool 2022]. Practical implementation may reveal additional factors that impact performance, such as hidden costs in intermediate operations. Furthermore, we propose studying its applicability to the corona product of other graph classes, such as permutation graphs, where isomorphism can be determined in linear time [Colbourn 1981].

Acknowledgment

The authors express their sincere gratitude to the Conselho Nacional de Desenvolvimento Científico (CNPq) for the financial support that enabled this research. This study was conducted as part of the activities developed under the Programa de Iniciação à Pesquisa (PIP-UFG).

References

Bhange, A. and Bhapkar, H. (2020). Perfect coloring of corona product of cycle graph with cycle, path and null graph. *Advances in Mathematics: Scientific Journal*, 9:10839–10844.

Bondy, J. A. and Murty, U. S. R. (2008). *Graph theory*. Springer Publishing Company, Incorporated.

Booth, K. S. and Colbourn, C. J. (1979). Problems polynomially equivalent to graph isomorphism. Technical Report CS-77-04, Department of Computer Science, University of Waterloo.

Colbourn, C. (1981). On testing isomorphism of permutation graphs. *Networks*, 11(1):13–21.

da Silva, D. and da Silva, H. (2022). Algoritmos para compor uma ferramenta web: geração de grafos grades e cordais e solução de conjunto independente, clique e emparelhamento em grafos. In *Anais da X Escola Regional de Informática de Goiás*, pages 118–129, Porto Alegre, RS, Brasil. SBC.

Feigenbaum, J. and Schaffer, A. (1985). Recognizing corona graphs. *AT&T Bell Laboratories Technical Memorandum*.

Frucht, R. and Harary, F. (1970). On the corona of two graphs. *Aequationes mathematicae*, 4(3):322–325.

Furmanczyk, H., Kubale, M., and Mkrtchyan, V. (2012). Equitable colorings of corona multiproducts of graphs. *Discussiones Mathematicae Graph Theory*, 37.

G. Yero, I., Kuziak, D., and Aguilar, A. (2012). Coloring, location and domination of corona graphs. *Aequationes Mathematicae*.

Graph problems tool (2022). Graph problems tool. `https://github.com/braully/graph-problems-tool`. Accessed on October 13, 2024.

Halim, S. (2015). Visualgo–visualising data structures and algorithms through animation. *Olympiads in informatics*, 9:243–245.

Sabidussi, G. (1959). The composition of graphs. *Duke Mathematical Journal*, 26(4):693 – 696.

Sharma, R., Adhikari, B., and Krueger, T. (2019). Self-organized corona graphs: A deterministic complex network model with hierarchical structure. *Advances in Complex Systems*, 22(06):1950019.

Silva, B. R. (2018). *Algoritmos e limites para os números envoltório e de Carathéodory na convexidade P3*. Dissertação (mestrado em ciência da computação), Universidade Federal de Goiás, Goiânia. 93 f.

UnickSoft (2024). Graph online. `https://graphonline.ru/en/`. Accessed on October 12, 2024.