Exact methods for the longest induced cycle problem

Ahmad Turki Anaqreh[1,*], **Boglárka Gazdag-Tóth**[1] and **Tamás Vinkó**[1]

[1] *University of Szeged, Institute of Informatics, 6720 Szeged, Árpád tér 2, Hungary*
E-mail: ⟨{ahmad, boglarka, tvinko}@inf.u-szeged.hu⟩

Abstract. The longest induced (or chordless) cycle problem is a graph problem classified as NP-complete and involves the task of determining the largest possible subset of vertices within a graph in such a way that the induced subgraph forms a cycle. Within this paper, we present three integer linear programs specifically formulated to yield optimal solutions for this problem. The branch-and-cut algorithm has been used for two models that cannot be directly solved by any MILP solver. To demonstrate the computational efficiency of these methods, we utilize them on a range of real-world graphs as well as random graphs. Additionally, we conduct a comparative analysis against approaches previously proposed in the literature.

Keywords: branch-and-cut algorithm, longest chordless cycle, longest induced cycle, mixed integer linear programming, valid inequalities

1. Introduction

A significant part of combinatorial optimization is closely related to graphs. Within graph theory, the concept of graph cycles has fundamental importance. Identifying a simple cycle or a cycle with a specific structure within a graph forms the basis for numerous graph-theoretical problems that have been under investigation for many years. One such problem is the Eulerian walk, a cyclic path that traverses each edge exactly once, as discussed in [19]. Another example is the Hamiltonian cycle, which traverses every vertex exactly once, as explored in [1].

Kumar et al. [11], introduced a heuristic algorithm for the longest simple cycle problem. The authors utilized both adjacency matrices and adjacency lists, achieving a time complexity for the proposed algorithm proportional to the number of nodes plus the number of edges of the graph. In [2], the authors investigated the longest cycle within a graph with a large minimal degree. For a graph $G = (V, E)$ with a vertex count of $|V| = n$, the parameter $\min\_\deg(G)$ denotes the smallest degree among all vertices in G, while $c(G)$ represents the size of the longest cycle within G. The authors demonstrated that for $n > k \geq 2$, with $\min\_\deg(G) \geq n/k$, the lower bound $c(G) \geq [n/(k-1)]$ holds. Broersma et al. [5] proposed exact algorithms for identifying the longest cycles in claw-free graphs. A claw, in this context, refers to a star graph including three edges. The authors introduced two algorithms for identifying the longest cycle within such graphs containing n vertices: one algorithm operates in $\mathcal{O}(1.6818^n)$ time with exponential space complexity, while the second algorithm functions in $\mathcal{O}(1.8878^n)$ time with polynomial space complexity.

In the work by Lokshtanov [12], the focus lies on the examination of the longest isometric cycle within a graph, which is defined as the longest cycle where the distance between any two

---

vertices on the cycle remains consistent with their distances in the original graph. The author introduced a polynomial-time algorithm to address this specific problem.

Our primary focus in this paper is dedicated to addressing the challenge of identifying the longest induced (or chordless) cycle problem. For a graph $G = (V, E)$ and a subset $W \subseteq V$, the W-induced graph $G[W]$ comprises all the vertices from set W and the edges from G that connect vertices exclusively within W. The objective of the longest induced cycle problem is to determine the largest possible subset W for which the graph $G[W]$ forms a cycle. While it may seem straightforward to obtain an induced cycle since every isometric cycle is an induced cycle, it has been shown that identifying the longest induced cycle within a graph is an NP-complete problem, as demonstrated by Garey et al. [8].

The longest induced path (P), discussed in [14], represents a sequence of vertices within graph G, where each consecutive pair of vertices is connected by an edge $e \in E$ and there is no edge between non-consecutive vertices within P. In the context of a general graph G, determining the existence of an induced path with a specific length is proven to be NP-complete, as detailed in [8]. Consequently, the longest induced cycle can be considered as a special case of the longest induced path.

Holes in a graph, defined as induced cycles with four or more vertices, play a significant role in various contexts. Perfect graphs, for instance, are characterized by the absence of odd holes or their complements [6]. Moreover, when addressing challenges like finding maximum independent sets in a graph [15], the existence of odd holes leads to the formulation of odd hole inequalities, strengthening approaches for these problems. Similarly, in other problem domains such as set packing and set partitioning [4], these odd hole inequalities serve as crucial components.

Several papers have explored the longest induced cycle problem in graphs with specific structures. In [7], the author investigated the longest induced cycle within the unit circulant graph. To define the unit circulant graph $X_n = Cay(\mathbb{Z}_n; \mathbb{Z}_n^*)$, where n is a positive integer, consider the following. The vertex set of X_n, denoted as $V(n)$, comprises the elements of $\mathbb{Z}_n$, the ring of integers modulo n. The edge set of X_n, represented as $E(n)$, for $x, y \in V(n)$, $(x, y) \in E(n)$ if and only if $x - y \in \mathbb{Z}_n^*$, with $\mathbb{Z}_n^*$ being the set of units within the ring $\mathbb{Z}_n$. The author demonstrates that if the positive integer n has r distinct prime divisors, then X_n contains an induced cycle of length $2^r + 2$. In a separate study by Wojciechowski et al. [20], the authors examine the longest induced cycles within hypercube graphs. If G represents a d-dimensional hypercube, they proved the existence of an induced cycle with a length $\geq (9/64) \cdot 2^d$.

Almost parallel to our work, Pereira et al. [17] dealt with the longest cordless cycle problem, which is equivalent to the longest induced cycle problem. They presented an integer linear program (ILP) formulation along with additional valid inequalities to strengthen and refine the formulation, all of which were incorporated into a branch-and-cut algorithm. They applied a multi-start heuristic method for initial solution generation and then conducted performance evaluations of the algorithm on a range of randomly generated graphs, including those with up to 100 vertices. They could solve the largest problems within 3,011.17 seconds. Our aim is to provide models and methods that can work more efficiently. The models and the best branch-and-cut versions proposed in [17] are discussed in Section 2.5.

Our paper proposes three integer linear programs (ILPs) designed to handle the longest induced cycle problem within general graphs. Some of these models were built based on those used in previous work focused on solving the longest induced path problem, as seen in the studies by Marzo et al. [13] and Bokler et al. [3]. Matsypura et al. [14] introduced three integer programming (IP) formulations and an exact iterative algorithm based on these IP formulations for tackling the longest induced path problem. However, it is important to note that we do not extend these methods, as they were found to be less effective compared to models in [13, 3].

The rest of the paper is organized as follows. Sections 2 and 3 discuss the models and methodologies used to solve the problem together with the best models and methods presented

in [17]. Section 4 reports the numerical results to show the efficiency of our models, also compared to the results in [17]. The conclusion of our work is presented in Section 5.

2. Models

2.1. Notations

Let $G = (V, E)$ be a directed graph with vertex set V and edge set $E \subset V \times V$. An edge $e = (i, j) \in E$ is defined for some $i, j \in V$. The symmetric pair is given as $\bar{e} = (j, i)$. For undirected graphs, we introduce the symmetric edge set $E^* = E \cup \{\bar{e} = (j, i) : e = (i, j) \in E\}$.

We use the notation δ for adjacent edges over vertices and edges as follows. Let us denote the outgoing and incoming edges incident to vertex i with $\delta^+(i) = \{(i, k) \in E^*\}$, and $\delta^-(i) = \{(k, i) \in E^*\}$, respectively. Additionally, $\delta(i) = \delta^+(i) \cup \delta^-(i)$ denote all the edges incident to vertex i.

For an edge $e = (i, j) \in E^*$, outgoing edges are $\delta^+(e) = \delta^+(i) \cup \delta^+(j) \setminus \{e, \bar{e}\}$, and similarly, incoming edges are $\delta^-(e) = \delta^-(i) \cup \delta^-(j) \setminus \{e, \bar{e}\}$. The neighbour edges of e are denoted by $\delta(e) = \delta^+(e) \cup \delta^-(e)$ for all $e \in E^*$. This notation can be extended to any subset of vertices $C \subset V$, where $\delta^+(C) := \{(k, l) \in E^* : k \in C, l \in V \setminus C\}$ and $\delta^-(C) := \{(k, l) \in E^* : l \in C, k \in V \setminus C\}$ denote the outgoing and incoming edges of C, respectively, and $\delta(C) = \delta^+(C) \cup \delta^-(C)$ all edges that connect C with $V \setminus C$.

2.2. Order-based model

The first model to discuss, called *LIC*, is an MILP model using order-based formulation to avoid subtours. The formalism of the model is as follows:

$$\max \sum_{i \in V} y_i \tag{1}$$

subject to

$$x_e + x_{\bar{e}} \leq 1 \qquad \forall\, e \in E \tag{2}$$

$$\sum_{g \in \delta^+(e)} x_g \leq 1 \qquad \forall\, e \in E^* \tag{3}$$

$$y_i = \sum_{g \in \delta^+(i)} x_g \qquad \forall i \in V \tag{4}$$

$$y_i = \sum_{g \in \delta^-(i)} x_g \qquad \forall i \in V \tag{5}$$

$$\sum_{i \in V} w_i = 1 \tag{6}$$

$$w_i \leq y_i \qquad \forall\, i \in V \tag{7}$$

$$u_i - u_j \leq n(1 - x_e) - 1 + nw_i \quad \forall e \in E^* \tag{8}$$

$$\sum_{i \in V} i w_i \leq j y_j + n(1 - y_j) \qquad \forall\, j \in V \tag{9}$$

$$y_i, u_i \geq 0 \qquad \forall i \in V \tag{10}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E^* \tag{11}$$

$$w_i \in \{0, 1\} \qquad \forall i \in V \tag{12}$$

The variable y_i indicates whether vertex i is part of the longest induced cycle or not. Consequently, the objective in (1) aims to maximize the sum of these variables, which directly corresponds to the length of the cycle. The decision variable x_e is one if the edge e is included in the solution, and zero otherwise.

The constraints can be understood as follows. Given that E^* is symmetric, constraint (2) guarantees that only one of the edges e or $\bar{e}$ can exist in the cycle, preventing the formation of small cycles. Constraint (3) ensures that for any edge $e = (i, j) \in E^*$, only one outgoing edge from either vertex i or vertex j can be part of the cycle. Constraints (4) and (5) ensure that for a given vertex i, only one outgoing edge and one incoming edge can be chosen to be part of the cycle. The variable w_i is introduced to handle the position of the last vertex in the solution, helping in the ordering process. Constraints (6)-(7) ensure that w_i is 1 for exactly one vertex and that w_i can be 1 only for one of the vertices in the solution, respectively. The constraint (8) is a modified Miller-Tucker-Zemlin (MTZ) order-based formulation: if edge $e = (i, j)$ is in the cycle, vertices i and j must be arranged in sequential order according to variables u_i and u_j such that $u_j \geq u_i + 1$, unless the binary variable w_i equals 1. Constraint (9) functions as a symmetry-breaking constraint, as described in [18]. It enforces that the last vertex in the cycle must have the smallest index among all vertices in the cycle.

For a variation of the above introduced LIC model consider the following constraint:

$$x_e + x_{\bar{e}} \geq y_i + y_j - 1 \qquad \forall\, e = (i, j) \in E \tag{13}$$

Constraint (13) guarantees that either edge e or $\bar{e}$ must be included in the solution if both endpoints i and j are part of the solution. Conversely, if an edge is not selected for the solution, neither of its endpoints can be included in the solution. By substituting constraint (3) in the original LIC model with constraint (13), we create a new model, $LIC2$. This modification leads to improved runtime performance compared to LIC, as demonstrated in Section 4.

2.3. Subtour-elimination model

The second model we employ to address the longest induced cycle problem is based on the model presented by Bokler et al. [3], which is referred to ILP_{cut} and was originally designed for identifying the longest induced path. E^* is the symmetric edge set, as defined previously. Let $\mathcal{C}$ represent the set of cycles in G. The model is defined as follows:

$$\max \frac{1}{2} \sum_{e \in E^*} x_e \tag{14}$$

subject to

$$x_e = x_{\bar{e}} \qquad \forall e \in E \tag{15}$$

$$x_e \leq \sum_{g \in \delta^-(i)} x_g \qquad \forall\, e = (i, j) \in E^* \tag{16}$$

$$\sum_{g \in \delta^-(i)} x_g + \sum_{g \in \delta^+(j)} x_g \leq 2 \qquad \forall\, e = (i, j) \in E^* \tag{17}$$

$$\sum_{e \in \delta(i)} x_e \leq \sum_{g \in \delta(C)} x_g \qquad \forall C \in \mathcal{C}, i \in C \tag{18}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E^* \tag{19}$$

The binary decision variable x_e indicates whether edge e is a part of the longest induced cycle, but unlike in the LIC model (in Section 2.2), in this case, edge selection is symmetric.

Consequently, the objective is to maximize half of the sum of these variables, as defined in objective function (14). Symmetry of the solution is guaranteed by (15). Constraint (16) enforces that the solution forms a cycle or cycles, while constraint (17) specifies that for any edge e in the solution, precisely two of its adjacent edges must also be in the solution. This ensures the induced property of the solution. Constraint (18) is utilized to eliminate small cycles in the graph.

2.4. Cycle-elimination model

Our third model, called *cec*, is a modified version of the *cec* model introduced in [13] to find the longest induced path. In this model, the symmetry of the edges is not used. The formalism of the model is as follows:

$$\max \sum_{i \in V} y_i \tag{20}$$

subject to

$$\sum_{e \in \delta(i)} x_e = 2y_i \qquad \forall\, i \in V \tag{21}$$

$$x_e \leq y_i \qquad \forall\, i \in V, e \in \delta(i) \tag{22}$$

$$x_e \geq y_i + y_j - 1 \quad \forall e = (i,j) \in E \tag{23}$$

$$\sum_{i \in C} y_i \leq |C| - 1 \qquad \forall C \in \mathcal{C} \tag{24}$$

$$y_i \in \{0,1\} \qquad \forall i \in V \tag{25}$$

$$x_e \in \{0,1\} \qquad \forall e \in E \tag{26}$$

The binary decision variable y_i maintains its previous interpretation, equal to 1 if vertex i is part of the solution. Additionally, variable x_e is set to 1 if edge e is included in the solution. However, in this context, the symmetric edge is not needed. The objective function (20) seeks to maximize the number of vertices within the induced cycle. Constraint (21) guarantees that each vertex within the solution is connected to precisely two vertices in the cycle. Constraints (22) and (23) are in place to ensure that the cycle is induced. To eliminate solutions composed of small cycles from consideration, constraint (24) is introduced. $\mathcal{C}$ represents a set of the cycles for the given graph. Constraint (24) is added to the model to enforce the solution to consist of a single cycle. This means that multiple small cycles are not deemed valid solutions.

2.5. Cordless-cycle model

The CCP formulation was introduced by Pereira et al. [17] to deal with the problem at hand. The CCP formulation is formally described as follows:

$$\max \sum_{i \in V} y_i \tag{27}$$

subject to

$$\sum_{e \in E} x_e = \sum_{i \in V} y_i \tag{28}$$

$$\sum_{i \in V} y_i \geq 4 \tag{29}$$

$$\sum_{e \in \delta(i)} x_e = 2y_i \qquad \forall\, i \in V \tag{30}$$

$$\sum_{g \in \delta(C)} x_g \geq 2(y_i + y_j - 1) \quad C \subset V, i \in C, j \in V \setminus C \tag{31}$$

$$x_e \leq y_i \qquad \forall\, i \in V, e \in \delta(i) \tag{32}$$

$$x_e \geq y_i + y_j - 1 \qquad \forall\, e = (i,j) \in E \tag{33}$$

$$x_e \in \{0,1\} \qquad \forall e \in E \tag{34}$$

$$y_i \in \{0,1\} \qquad \forall i \in V \tag{35}$$

The formulation includes the usual sets of binary variables: y_i and x_e, indicating whether vertex i and edge e are in the cycle or not. Consequently, the number of selected vertices and edges is equal, as required by (28), and at least four vertices must be selected by (29). Each vertex within the solution is incident to precisely two edges, as guaranteed by (30). Moreover, the subgraph defined by x and y remains connected, as guaranteed by the subtour elimination constraint (31). Furthermore, (32)–(33) ensures that any solution is an induced subgraph of G. More specifically, any edge of G with both its endpoints belonging to the solution must be part of the solution.

The CCP formulation was employed by the authors of [17], along with various valid inequalities. They introduced nine branch-and-cut (BC) algorithms and subsequently chose the top three among them. The first one, labeled as $BC1$ contains constraints (28)-(35), and in addition the following constraint:

$$\sum_{g \in \delta(C)} x_g \geq 2x_e \quad C \subset V, e = (i,j) \in E, i \in C, j \in V \setminus C \tag{36}$$

This algorithm initiates by separating (31), and subsequently, the resulting inequality is enhanced to the more robust form of (36). This specific constraint ensures that if $x_e = 1$, then it is mandatory for $y_i = y_j = 1$ to hold true, due to the presence of inequalities (32)-(33). For the $BC2$ and $BC3$ algorithms, both constraints (37) and (38) were included together with constraints (28)-(36).

$$\sum_{i \in Q} y_i \leq 2 \tag{37}$$

$$\sum_{e \in E(Q)} x_e \geq \sum_{i \in Q} y_i - 1 \tag{38}$$

For a clique $Q \subset V, |Q| \geq 3$. Constraint (37) ensures that within a clique Q at most two of its vertices can be part of the induced cycle. On the other hand, constraint (38) guarantees that for a clique Q the number of vertices that can be part of the induced cycle is limited to at most one more than the number of edges that can be included from the clique. Namely, only one of the edges from Q might be included in the solution.

For the $BC2$ algorithm, they implemented a rule that imposes no restrictions on the number of separation rounds. In other words, whenever a violated inequality is detected, it is included

in the cut pool. Conversely, for $BC3$, a fixed number of separation rounds, specifically 30, was established, and inequalities were added to the cut pool if a clique did not share two or more vertices with a clique in a previously accepted inequality. The order of inequalities in the cut pool was determined by descending order of the absolute values of their corresponding linear programming relaxation dual variables. All three algorithms utilized the lower bounds obtained from the multi-start CCP heuristic [17], which is a constructive procedure that takes a predefined edge as input data. The algorithm then seeks to extend a tentative path, P, containing the selected vertices. Vertices are added to P one at a time, accepted if they are adjacent to one of the path's current extremities and not adjacent to any internal vertices. The procedure terminates when the endpoints of P meet, resulting in a chordless cycle of G, or when further expansion of P becomes impossible.

3. Algorithms

Out of the three models we have introduced, only LIC (and $LIC2$) can be directly solved using any MILP solver. Both ILP_{cut} and cec rely on the set of small cycles, which are usually created as part of the solution process, either through an iterative cut generation approach or, more effectively, via branch-and-cut algorithm by employing separation.

Note that subtour elimination inequalities (18) and (24), present in the ILP_{cut} and cec models respectively, exhibit exponential complexity. Consequently, attempting to enumerate all inequalities corresponding to each subtour within the graph and subsequently cutting them becomes impractical. Instead, we have added these inequalities to the ILP_{cut} and cec models as soon as facing them. Hence, the cut generation approach is employed as follows: the method is initiated with a model relaxing all subtour elimination inequalities, and if subtours arise in integer solutions, violated inequalities are added, and this process is repeated until the optimal solution is reached. For that, callback functionality from Gurobi [9] was employed, which can be used to add these inequalities iteratively.

We employed the Depth-First Search (DFS) algorithm on the induced subgraph of the integer solution to identify cycles, subsequently introducing a new inequality for each subtour discovered.The entire procedure, which combines the models and cut generations, is shown in Algorithms 1 and 2.

3.1. Initialization

In the initialization phase of the procedure, the ILP_{cut} and cec models are created, encompassing the creation of their variables, constraints, and objective functions.

3.2. Cut generation

To tackle the ILP_{cut} and cec models, we combine the cut generation mechanism with the Branch-and-bound method, as explained earlier in this section. Consequently, each model was addressed using two distinct methodologies, as outlined below.

3.2.1. Soft approach

The first approach involves cut generation as outlined in Algorithm 1. In each iteration, a subproblem from the Branch-and-Bound tree is solved. In line 4 the algorithm checks if the solution of the subproblem is an integer solution. Based on this, the DFS algorithm is employed to detect any subtours within the solution, as shown in line 5. If a subtour exists, and its length is less than or equal to the value of the variable `longest_induced_cycle`, a cut is appended for that cycle. If not, the value of the variable is updated to reflect the length of the cycle, and

there is no need to introduce a cut because the cycle could potentially be the optimal solution. These details are clarified in lines 6 through 9. The cut generation terminates when there are no further subtours present in the solution, indicating the completion of the procedure.

Algorithm 1 Cut_Generation1

1: **Input:** model Initialization()
2: `longest_induced_cycle` ← 0
3: **function** Cut_Generation1()
4:    **if** `model.status == feasible_integer` **then**   ▷ model has integer solution
5:       `C` ← DFS(`feasible_integer`)   ▷ find subtour in the solution
6:       **if** length(`C`) ≤ `longest_induced_cycle` **then**
7:          `model.addConstr(18, 24)`   ▷ add cut (18) or (24) to the model
8:       **else**
9:          `longest_induced_cycle` ← length(`C`)   ▷ update variable value
10:       **end if**
11:    **end if**
12: **end function**
13: `model.optimize(Cut_Generation1())`   ▷ solve the model using cut generation
14: `print(longest_induced_cycle)`

3.2.2. Tough approach

The second cut generation-based approach is detailed in Algorithm 2. In each iteration, a subproblem is solved, and if an integer solution is obtained, the algorithm verifies the presence of any subtours using the DFS algorithm, as described in lines 4 through 5. If any cycles are detected, a cut is integrated into the model (line 6), and the length of the cycle is updated if it exceeds the value of the variable `longest_induced_cycle` (line 8). Although we may cut the optimal induced cycle, its length (and possibly the cycle itself) is recorded. It is important to note that in order to further improve the procedure, a constraint is added to the model in line 10. This constraint ensures that the objective value must be greater than the length of the largest induced cycle discovered so far. By using this cut generation, the branch-and-cut indicates the infeasibility of the problem, yet the longest induced cycle length recorded in the variable `longest_induced_cycle`.

Algorithm 2 Cut_Generation2

1: **Input:** model Initialization()
2: `longest_induced_cycle` ← 0
3: **function** Cut_Generation2()
4:    **if** `model.status == feasible_integer` **then**   ▷ model has integer solution
5:       `C` ← DFS(`feasible_integer`)   ▷ find subtour in the solution
6:       `model.addConstr(18, 24)`   ▷ add cut (18) or (24) to the model
7:       **if** length(`C`) > `longest_induced_cycle` **then**
8:          `longest_induced_cycle` ← length(`C`)
9:       **end if**
10:       `model.addConstr(model.ObjVal >= longest_induced_cycle + 1)`
11:    **end if**
12: **end function**
13: `model.optimize(Cut_Generation2())`   ▷ solve the model using cut generation
14: `print(longest_induced_cycle)`

3.3. Longest Isometric Cycle

Lokshtanov's algorithm, as described in [12], aims to identify the longest isometric cycle within a graph. In accordance with the definition of an isometric cycle, as discussed in Section 1, if a given graph G contains an isometric cycle with a length of ℓ, then there must also exist an induced cycle within the graph with a length of m where $m \geq \ell$. Consequently, the longest isometric cycle serves as a benchmark for the longest induced cycle. The algorithm's objective is to verify the existence of an isometric cycle with a length of k in a given graph $G = (V, E)$. If such a cycle exists, the graph G can be employed to construct a new graph G_k with vertices as vertex-pairs of G. Namely, $V(G_k) = \{(u, v) \in V : d(u, v) = \lfloor k/2 \rfloor\}$, where $d(u, v)$ is the length of the shortest path between u and v, and its edge set given by $E(G_k) = \{((u, v), (w, x)) : (u, w) \in E(G) \wedge (v, x) \in E(G)\}$.

The method is outlined in Algorithm 3. For a given value of k, the algorithm computes the graph G_k and examines whether there exists a pair of vertices (u, v) and (v, x) within $V(G_k)$ such that (v, x) belongs to the set $M_k(u, v) := \{(u, x) : (u, x) \in V(G_k) \wedge (v, x) \in E(G)\}$ and $d_{G_k}[(u, v), (v, x)] = \lfloor k/2 \rfloor$. If such a pair is found, it indicates the presence of an isometric cycle with a length of k.

Algorithm 3 Longest Isometric Cycle

1: LISC $\leftarrow 0$
2: **for all** $l \in V, i \in V, j \in V$ **do**   ▷ distance calculation by Floyd algorithm
3:     $d(i, j) \leftarrow \min\{d(i, j), d(i, l) + d(j, l)\}$
4: **end for**
5: **if** G is a tree **then**   ▷ no cycles in tree graph
6:     **return** LISC
7: **end if**
8: **for** $k = 3 \rightarrow n$ **do**
9:     $V_k \leftarrow \emptyset$ ▷ vertices of G_k
10:     **for all** $u, v \in V$ **do**
11:         **if** $d(u, v) = \lfloor k/2 \rfloor$ **then**
12:             $V_k \leftarrow V_k \cup \{(u, v)\}$
13:         **end if**
14:     **end for**
15:     $E_k \leftarrow \emptyset$ ▷ edges of G_k
16:     **for all** $(u, v), (w, x) \in V_k$ **do**
17:         **if** $(u, w) \in E \wedge (v, x) \in E$ **then**
18:             $E_k \leftarrow E_k \cup \{((u, v), (w, x))\}$
19:         **end if**
20:     **end for**
21:     $G_k \leftarrow (V_k, E_k)$
22:     **for all** $(u, v, x) \in V$ **do**
23:         **if** $(u, v) \in V_k \wedge (v, x) \in M_k(v, u) \wedge d_{G_k}[(u, v), (v, x)] = \lfloor k/2 \rfloor$ **then**
24:             LISC $\leftarrow k$
25:         **end if**
26:     **end for**
27: **end for**
28: print(LISC)

---

4. Numerical experiments

To demonstrate and evaluate the effectiveness of the proposed methods, we present numerical results for three models: the *LIC* model, the *ILP*$_{cut}$ model, and the *cec* model. Furthermore, we conducted a comparison between our best results and results from [17] on randomly generated graphs to highlight the efficiency of our approach in comparison to existing methods.

4.1. Computational environment and datasets

The algorithms detailed in Section 3 were implemented in Julia 1.7.0, utilizing the JuMP package version 0.22.1. We employed Gurobi 9.5.0 as the solver for all experiments. Each run was constrained to a one-hour time limit and a single thread. For the longest isometric cycle algorithm, we implemented it using Python 3.8 with a 24-hour time limit. These computations were performed on a computer with an Intel Core i7-4600U CPU, 8GB of RAM, and running the Windows 10 operating system.

To verify the efficacy of our methods, we employed two sets of network datasets. The first is the RWC set, comprising 19 real-world networks that encompass communication and social networks within companies, networks of book characters, as well as transportation, biological, and engineering networks, as described in [14]. Additionally, we utilized the Movie Galaxy (MG) set, consisting of 773 graphs that represent social networks among movie characters, as detailed in [10]. For further information about these instances, the reader is referred to the following link: http://tcs.uos.de/research/lip.

To perform a comparison with the results presented in [17], we conducted experiments on random graphs with varying values of n ranging from 50 to 100, considering both 10% and 30% density, as in [17]. For every case, 10 graphs were generated. Every run was restricted to a maximum duration of one hour, with no restrictions on the number of threads, and with an initial solution set to 4, as described in [17]. Regarding the hardware comparison, we utilized the information available in [16] to collect the details of the CPU utilized in all experiments, as outlined in Table 1. It is evident that the computer used in [17] is more powerful than ours. To ensure a fair comparison, we normalized the execution times in all cases. The ratio between the single-thread ratings gives a good approximation of the relative speed. Therefore, we calculated this ratio in the last row of Table 1. The run time was then modified by multiplying it by the obtained ratio.

Benchmarks	Intel Core i7-4600U	Intel Xeon W-3223 [17]
Clock Speed (GHz)	2.1	3.5
Turbo Speed (GHz)	Up to 3.3	Up to 4.0 GHz
Number of Physical Cores	2 (Threads: 4)	8 (Threads: 16)
Single Thread Rating	1641	2480
Ratio	0.66	1

Table 1: *CPU performance comparison between the CPU used in this paper and in [17].*

4.2. Computational results

Table 2 presents the computational experiments conducted on the RWC instances. The second column displays the optimal solutions for each instance (opt). In the third column, we find the length of the longest isometric cycle (*LISC*), if possible. The fourth and fifth columns respectively indicate the number of vertices (N) and edges (M) of the corresponding graph.

Columns six through eleven show the time in seconds required to identify the optimal solution using the various methods employed in this study. Specifically, $ILP_{cut}2$ and $cec2$ refer to the methods outlined in Algorithm 2. For all these methods, we also initiated the search using the $LISC$ value, incorporating the constraint `ObjVal` $\geq$ `LISC`. These methods are indicated in every second row corresponding to each graph. Instances that resulted in timeouts are denoted by the symbol ⏱.

graph	opt	LISC	N	M	LIC	$LIC2$	ILP_{cut}	$ILP_{cut}2$	cec	$cec2$
high-tech	10	5	33	91	1.22	0.63	0.95	0.33	0.38	0.14
					0.99	0.42	1.15	1.57	1.02	0.77
karate	6	5	34	78	0.63	0.58	0.21	0.24	0.20	0.19
					0.66	0.53	0.23	0.29	0.24	0.17
mexican	13	7	35	117	0.92	0.82	0.66	0.88	0.24	0.20
					0.83	0.78	0.69	0.74	0.20	0.20
sawmill	6	5	36	62	0.54	0.43	**0.18**	0.37	**0.15**	0.10
					0.33	0.30	**0.36**	0.16	**0.13**	0.11
tailorS1	12	7	39	158	2.93	1.11	1.37	1.45	0.34	0.33
					1.38	0.89	1.76	1.76	0.37	0.44
chesapeake	15	5	39	170	1.01	0.69	0.56	0.81	0.24	0.22
					1.05	0.72	0.97	0.87	0.28	0.31
tailorS2	12	5	39	223	3.11	2.05	3.49	4.46	0.74	0.65
					3.25	3.09	3.74	4.62	0.83	0.75
attiro	28	9	59	128	0.93	1.24	0.55	0.53	0.18	0.24
					0.67	0.71	0.52	0.68	0.28	0.31
krebs	8	7	62	153	10.91	7.23	**1.19**	0.94	**0.94**	0.48
					10.39	5.56	**0.86**	1.02	**0.57**	0.38
dolphins	20	7	62	159	14.75	23.70	1.81	2.35	1.70	1.02
					10.66	13.83	2.80	2.98	0.74	1.50
prison	28	9	67	142	5.90	10.22	**0.83**	1.56	**0.62**	0.61
					6.93	10.23	**4.81**	1.03	**0.66**	0.48
huck	5	5	69	297	519.95	299.12	**19.53**	17.79	**4.31**	4.51
					447.72	493.74	**18.22**	19.71	**3.34**	3.31
sanjuansur	35	11	75	144	6.16	5.85	0.68	0.71	0.37	0.49
					7.70	3.61	0.82	1.36	0.44	0.38
jean	7	5	77	254	276.98	147.77	15.93	14.57	2.45	2.41
					150	147.85	13.97	14.80	2.32	2.41
david	15	8	87	406	544.99	308.06	46.23	54.23	5.22	4.36
					219.14	323.96	45.24	38.33	3.31	3.47
ieeebus	32	13	118	179	2.52	5.14	0.76	0.94	0.43	0.62
					7.82	5.82	0.79	1.35	0.33	0.30
sfi	3	3	118	200	6.98	6.51	**0.74**	0.90	**0.3**	0.31
					6.40	2.80	**0.84**	1.32	**0.34**	0.31
anna	15	⏱	138	493	90.75	52.11	10.60	23.65	1.37	1.71
					-	-	-	-	-	-
494bus	116	⏱	494	586	108.48	126.77	27.13	33.09	2.73	2.10
					-	-	-	-	-	-
average					84.19	52.63	7.02	8.41	1.21	1.09
					51.52	59.70	5.75	5.45	0.9	0.92

Table 2: *Running times on RWC instances, time is given in seconds.*

The various methods exhibit diverse performance characteristics in terms of execution time and the number of instances solved optimally. Key observations from Table 2 are as follows:

- $cec2$ outperforms cec in 13 cases, ILP_{cut}, $ILP_{cut}2$, LIC and $LIC2$ in all the cases.

- $ILP_{cut}2$ was faster than $LIC2$ for 15 cases and LIC for all instances.

- $LIC2$ outperforms LIC in 14 cases.

- For some instances in ILP_{cut} and cec, the graphs and results are indicated by boldface and underlined in Table 2. This is to emphasize that these graphs contain multiple longest induced cycles of the same length, and the procedures described in Algorithm 1 cut the cycle if its length is less than or equal to the longest induced cycle found so far. Thus, for these graphs, all the longest cycles are found by the method.

- Using $LISC$ as an initial solution does not contribute significantly to improving the execution time in the majority of cases.

- The results emphasize the correlation between graph density and execution time. Graph density is defined as the ratio of the edges present in a graph to the maximum number of edges it can hold. This relationship is particularly evident for dense graphs like huck, jean, and david, especially in the case of LIC and $LIC2$ models. However, it is not the case for cec and $cec2$ models as their running times show less sensitivity to the graph's density.

nr. of edges	nr. of instances	LIC	$LIC2$	ILP_{cut}	$ILP_{cut}2$	cec	$cec2$
1–49	107	0.14	0.14	0.12	0.18	0.09	0.08
50–74	135	0.37	0.29	0.2	0.3	0.17	0.12
75–99	151	0.7	0.58	0.32	0.39	0.22	0.17
100–124	121	1.74	1.32	0.51	0.65	0.29	0.24
125–149	90	4.5	3.67	0.82	0.99	0.37	0.34
150–199	89	10.45	7.48	1.49	1.7	0.56	0.49
200–629	80	169.38	110.49	13.28	16.39	2.41	1.9
average		157.23	126.35	44.73	57.28	26.84	22.01

Table 3: *Running times on MG instances, time is given in seconds.*

The results for the MG instances, organized into groups based on the number of edges, are presented in Table 3. Unlike LIC and $LIC2$, where the running times increase proportionally with the instance size, the results indicate that cec and $cec2$ are more reliable, with running times showing less sensitivity to the graph's size.

The results for the random graphs are presented in Table 4, where we compare $cec2$ against the top three algorithms introduced in [17]. The runtime represents the average duration on the ten graphs in each case. Notably, $cec2$ outperforms these algorithms in all cases, even before normalizing the execution times with the ratio listed in Table 1. Moreover, $cec2$ successfully solved the instance with 100 vertices and 30% density, a scenario where none of the algorithms in [17] succeeded.

Randomly generated graphs: 10% density				
n	$cec2$	$BC1$	$BC2$	$BC3$
50	0.32 (0.49)	0.33	0.3	0.39
60	0.79 (1.2)	1.19	1.2	1.34
70	4.17 (6.32)	5.57	4.93	5.58
80	20.5 (31.1)	37.15	26.9	27.34
90	93.93 (142.32)	160.1	155.82	168.02
100	518.75 (785.99)	1321.41	1129.47	1094.8
average	106.41	254.29	219.77	216.25
Randomly generated graphs: 30% density				
n	$cec2$	$BC1$	$BC2$	$BC3$
50	4.15 (6.28)	8.21	9.6	8.78
60	26.14 (39.6)	39.82	46.18	51.9
70	90.11 (136.52)	234.45	206.36	283.28
80	203.81 (308.8)	935.78	676.52	1139.55
90	544.88 (825.58)	2072.16	1874.91	3011.17
100	1810.49 (2743.17)	⏱	⏱	⏱
average	446.6	658.08	562.71	898.94

Table 4: *Running times on random instances for* cec*2,* BC*1,* BC*2, and* BC*3, time is given in seconds. Values in brackets show the original execution time of cec2.*

5. Conclusion

Considering that the longest induced cycle problem is NP-hard, it is essential to find an efficient approach that can yield optimal solutions within a reasonable time. In this regard, we introduced three integer linear programs, some of which are extensions of models originally formulated for solving the longest induced path problem. These newly proposed programs showed differing execution times and success rates in terms of achieving optimal solutions, outperforming the models presented in the literature. We have found that the $cec2$ formulation with tough cut generation yields the most efficient method. For future work, heuristic or metaheuristic approaches can be used as initial solutions for the MILP, potentially increasing the efficiency of solving the problem. Designing more sophisticated MILP formulations with constraints that can more effectively prune the search space is another possibility. Additionally, exploring exact methods like branch-and-price algorithms, which combine branch-and-bound with column generation, could be effective in handling large and complex instances.

References

[1] Akiyama, T. and Nishizeki, T. and Saito, N. (1980). NP-completeness of the Hamiltonian cycle problem for bipartite graphs. Journal of Information Processing, 3(2), 73-76.

[2] Alon, N. (1986). The longest cycle of a graph with a large minimal degree. Journal of Graph Theory, 10(1), 123-127. doi: 10.1002/jgt.3190100115

[3] Bökler F., Chimani M., Wagner M. H. and Wiedera T. (2020). An experimental study of ILP formulations for the longest induced path problem. International Symposium on Combinatorial Optimization, 89-101. doi: 10.1007/978-3-030-53262-8_8

[4] Borndörfer, R. (1998). Aspects of set packing, partitioning, and covering. PhD thesis, TU Berlin, Berlin.

[5] Broersma, H., Fomin, F. V., van't Hof, P. and Paulusma, D. (2013). Exact algorithms for finding longest cycles in claw-free graphs. Algorithmica, 65(1), 129-145. doi: 10.1007/s00453-011-9576-4

[6] Chen, Y. and Flum, J. (2007). On parameterized path and chordless path problems. Twenty-Second Annual IEEE Conference on Computational Complexity (CCC'07), 250–263. doi: 10.1109/CCC.2007.21

[7] Fuchs, E. D. (2005). Longest induced cycles in circulant graphs. The Electronic Journal of Combinatorics, Article Number R52. doi: 10.37236/1949

[8] Garey, M. R. and Johnson, D. S. (1990). Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co.

[9] Gurobi Optimization Inc. Gurobi Optimizer Reference Manual (2020).

[10] Kaminski, J., Schober, M., Albaladejo, R., Zastupailo, O. and Hidalgo, C. (2018). Moviegalaxies-social networks in movies. Retrieved from: cris.maastrichtuniversity.nl

[11] Kumar, P. and Gupta, N. (2014). A heuristic algorithm for longest simple cycle problem. Proceedings of the International Conference on Wireless Networks (ICWN), 202-208.

[12] Lokshtanov, D. (2009). Finding the longest isometric cycle in a graph. Discrete Applied Mathematics, 157(12), 2670-2674. doi: 10.1016/j.dam.2008.08.008

[13] Marzo, R., Melo, R. A., Ribeiro, C. C. and Santos, M. C. (2022). New formulations and branch-and-cut procedures for the longest induced path problem. Computers & Operations Research, 139, 105627. doi:10.1016/j.cor.2021.105627

[14] Matsypura, D., Veremyev, A., Prokopyev, O. A. and Pasiliao, E. L. (2019). On exact solution approaches for the longest induced path problem. European Journal of Operational Research, 278(2), 546-562. doi: 10.1016/j.ejor.2019.04.011

[15] Nemhauser, G. L. and Sigismondi, G. (1992). A strong cutting plane/branch-and-bound algorithm for node packing. Journal of the Operational Research Society, 43(5), 443-457. doi: 10.1057/jors.1992.71

[16] PassMark Software - CPU Benchmarks (2023). Retrieved from: cpubenchmark.net/compare

[17] Pereira, D. L., Lucena, A., Salles da Cunha, A. and Simonetti, L. (2022). Exact solution algorithms for the chordless cycle problem. INFORMS Journal on Computing, 34(4),1970-1986. doi: 10.1287/ijoc.2022.1164

[18] Walsh, T. (2012). Symmetry breaking constraints: Recent results. Proceedings of the AAAI Conference on Artificial Intelligence, 26(1), 2192-2198. doi: 10.1609/aaai.v26i1.8437

[19] West, D. B. (2001). Introduction to Graph Theory. Prentice Hall.

[20] Wojciechowski, J. M. (1990). Long induced cycles in the hypercube and colourings of graphs. University of Cambridge.