RAIRO-Oper. Res. 55 (2021) 333–353
https://doi.org/10.1051/ro/2021004

RAIRO Operations Research
www.rairo-ro.org

EXACT AND APPROXIMATE ALGORITHMS FOR THE LONGEST INDUCED PATH PROBLEM

Ruslán G. Marzo and Celso C. Ribeiro*

Abstract. The longest induced path problem consists in finding a maximum subset of vertices of a graph such that it induces a simple path. We propose a new exact enumerative algorithm that solves problems with up to 138 vertices and 493 edges and a heuristic for larger problems. Detailed computational experiments compare the results obtained by the new algorithms with other approaches in the literature and investigate the characteristics of the optimal solutions.

Mathematics Subject Classification. 05C30, 05C38, 05C85, 68W25, 90C27, 90C35.

Received October 16, 2020. Accepted January 14, 2021.

1. Introduction

Let $G = (V, E)$ be an undirected graph defined by its set of n vertices $V = \{1, \cdots, n\}$ and its edge set $E \subseteq V \times V$. For any subset of vertices $S \subseteq V$, let $G[S] = (S, E')$ denote the subgraph induced by S in G, where E' contains all edges from E that have both of their endpoints in S. The *longest induced path problem* (LIPP) is defined as that of finding the subset $S \subseteq V$ of largest cardinality such that the resulting induced subgraph, $G[S]$, is a simple path [13].

The longest induced path problem has applications in various network analysis and design contexts. The graph diameter, which is defined as the length of the longest shortest path in a graph, is often used to quantify graph communication properties. In particular, the graph diameter provides an intuitive measure of the worst-case pairwise distance, *e.g.*, the longest of the shortest communication paths in a communication network or the longest of the shortest routes in a transportation network. The longest induced path can also be considered as a measure of the worst-case pairwise distance in other practical scenarios, where vertices can either fail, be overloaded or be destroyed by an adversary (depending on the application context) and thus cannot be used in any communication or transportation path. In these cases, some subset of vertices cannot transmit a message in a communication network or cannot serve as a transshipment point in a transportation network and, consequently, detours or alternative shortest paths must be used. In other words, the objective of the longest induced path problem is to identify the worst possible case for the shortest distance between any two vertices in the graph, given that these vertices remain connected by some path while the rest of the vertices may fail [13].

The longest induced path problem is also known in the literature as the maximum induced path problem (see *e.g.* [2, 7]). The maximum weighted induced path problem has applications in large communication and

Keywords. Longest induced path problem, exact algorithm, backtracking, heuristics.

Institute of Computing, Universidade Federal Fluminense, 24210-346 Niterói, RJ, Brazil.
*Corresponding author: celso@ic.uff.br

neural networks when the worst case communication time needs to be evaluated [7]. Furthermore, LIPP is related to the snake-in-the-box problem, which was described by Kautz [11] and has applications in the theory of error-correcting codes. It consists in finding the longest possible induced path along the edges of a hypercube graph.

Another interesting interpretation of LIPP appears in social networks with information cascades. Matsypura *et al.* [13] observed that an optimal solution of the longest induced path problem in this context provides the longest possible path of information transmission, where one end vertex of the path corresponds to the seed of the information cascade and the other end vertex is the last to learn this piece of information.

Even though the graph diameter is computable in polynomial time [1], the decision version of the longest induced path problem is NP-complete ([6], p. 196). The problem remains NP-complete even for bipartite graphs. Finding the longest simple path (not necessarily induced) is also NP-complete. Polynomially solvable classes of the problem for specific graph families are discussed *e.g.* in [7–10, 12]. Bergougnoux and Kanté [4] designed a framework to obtain efficient algorithms for several problems with a global constraint (acyclicity or connectivity), including the longest induced path.

Although the longest induced path problem belongs to an interesting class of network optimization problems with practical applications in various network contexts, the literature on solution approaches for this problem is rather limited, possibly due to its inherent computational complexity. Matsypura *et al.* [13] were the first to develop exact solution approaches for the longest induced path problem for general graphs, based on integer programming (IP) techniques. They proposed four IP formulations based on three conceptually different interpretations of the problem. In addition, they provided an exact iterative algorithm that solves a sequence of smaller MIPs and developed a randomized heuristic to find induced paths in large networks. Bökler *et al.* [5] proposed stronger integer linear programming formulations using cut (or generalized subtour elimination) constraints and clique inequalities.

In this work, we focus on solving the longest induced path problem for different classes of general graphs. We propose an exact enumerative algorithm and a heuristic for the problem. We review in Section 2 the existing solutions approaches proposed by Matsypura *et al.* [13] and Bökler *et al.* [5], from where we take the same test instances that will be used in our computational experiments. Section 3 describes the proposed exact enumerative algorithm and a new heuristic. In Section 4 we present computational experiments to assess the performance of the proposed solution methods. Section 5 investigates the correlation of the running times of the proposed exact algorithm and the sizes of the longest induced paths with some characteristics of the test instances. We also analyze the eccentricity and the degree of the extreme vertices of the optimal solutions obtained with this exact algorithm. Concluding remarks are drawn in Section 6.

2. Existing approaches

In this section, we first summarize two integer programming formulations for the longest induced path problem (LIPP) developed by Matsypura *et al.* [13], referred to as IP3 and IP3c. The key idea behind these formulations is first to model a walk in the underlying graph and then force it to be an induced path by using additional constraints.

We assume that a walk in a graph starts at some vertex at time $t = 0$. Then, at time $t = 1$, it visits one of its neighbors. The walk continues visiting other vertices until time $t = T$ or no further moves are possible. The value of T can be set to $|V| - 1$ unless a better upper bound on the size of the longest induced path is available.

Both models IP3 and IP3c make use of the same binary variable x_i^t associated to each vertex $i \in V$ of the graph and each time $t \in \{0, \cdots, T\}$:

$$x_i^t = \begin{cases} 1, & \text{if and only if vertex } i \text{ is visited at time } t, \\ 0, & \text{otherwise.} \end{cases}$$

A new vertex is visited at each time period whenever possible. If the walk cannot be continued at time $t = \tau$, then $x_i^t = 0$ for all $i \in V$, $t \geq \tau$. To ensure that the walk is an induced path, it is necessary to verify that there are no shortcuts to the previously visited vertices and that no vertex that is not a neighbor of the current vertex can be visited at the next step. Based on these considerations, formulation IP3 is:

$$\text{(IP3):} \qquad \max \sum_{t=0}^{T} \sum_{i \in V} x_i^t \tag{2.1}$$

subject to:

$$\sum_{i \in V} x_i^t \leq 1, \qquad \forall t \in \{0, \cdots, T\}, \tag{2.2}$$

$$\sum_{t=0}^{T} x_i^t \leq 1, \qquad \forall i \in V, \tag{2.3}$$

$$x_i^t + x_j^{t+1} \leq 1, \qquad \forall (i,j) \notin E, \forall t \in \{0, \cdots, T-1\}, \tag{2.4}$$

$$x_i^t + x_j^{\tau} \leq 1, \qquad \forall (i,j) \in E, \forall t \in \{0, \cdots, T-2\}, \forall \tau \in \{t+2, \cdots, T\}, \tag{2.5}$$

$$\sum_{i \in V} x_i^t \leq \sum_{i \in V} x_i^{t-1}, \qquad \forall t \in \{1, \cdots, T\}, \tag{2.6}$$

$$x_i^t \in \{0,1\}, \qquad \forall i \in V, \forall t \in \{0, \cdots, T\}. \tag{2.7}$$

The objective function (2.1) maximizes the number of vertices in the resulting walk. Constraints (2.2) enforce that only one vertex can be visited at any time. Constraints (2.3) ensure that the walk cannot visit any vertex twice. Constraints (2.4) state that the walk can traverse only existing edges in E. Constraints (2.5) ensure that there are no shortcuts in a walk, *i.e.*, the walk is, in fact, an induced path. Constraints (2.6) enforce that the walk cannot be interrupted, *i.e.*, if the walk stops at time $t-2$ and no vertices are visited at time $t-1$, then it cannot start again from another vertex at time t. Finally, constraints (2.7) represent the integrality requirements.

Formulation IP3 has $O(T|V|)$ variables and $O(T|V|^2 + T^2|E|)$ constraints. The number of constraints can be reduced by aggregating constraints (2.4) and (2.5), resulting in a more compact formulation denoted by IP3c with the same LP bound; see ([13], Sect. 3). Specifically, constraints (2.4) can be aggregated as:

$$\sum_{j \in V : (i,j) \notin E} x_j^{t+1} \leq 1 - x_i^t, \qquad \forall i \in V, \ \forall t \in \{0, \cdots, T-1\}, \tag{2.8}$$

meaning that if vertex i is visited at time t, *i.e.*, $x_i^t = 1$, then all vertices non-adjacent to i, *i.e.*, those in $\{j \in V \mid (i,j) \notin E\}$, cannot be visited at time $t+1$. On the other hand, if $x_i^t = 0$, then the above constraint is enforced due to constraint (2.2). Constraints (2.5) can be aggregated in a similar fashion:

$$\sum_{\tau=t+2}^{T} x_j^{\tau} \leq 1 - x_i^t, \qquad \forall (i,j) \in E, \ \forall t \in \{0, \cdots, T-2\}, \tag{2.9}$$

by observing that if a vertex i is visited at time t, then its neighbors cannot be visited at any time later than $t+1$. On the other hand, if $x_i^t = 0$, then the constraint is valid due to constraints (2.3).

The exact iterative algorithm of [13] solves a sequence of smaller IPs to obtain an optimal solution for the original problem. To ensure that formulations IP3 and IP3c are valid, we must set $T = |V| - 1$, unless a better upper bound for the longest induced path is known. If the length L^* of the induced path returned for some value of T is strictly smaller than T, then L^* is the optimal value for the original problem and the corresponding induced path is the longest one. Since there always exists an induced path whose length is equal to $\operatorname{diam}(G)$, the iterative search for the longest induced path may start from $T = \operatorname{diam}(G) + 1$. If $L^* \ll |V|$, then each IP is expected to be solved much faster than the original formulation with the full upper bound.

Matsypura *et al.* [13] also developed a randomized heuristic based on random walks. Its key idea consists in generating random walks that are also induced paths starting from each vertex $s \in S \subseteq V$, where S is some predefined subset of vertices. The incumbent is initialized with an empty path. For each vertex $s \in S$ the heuristic builds k random walks from s, where k is a parameter. The longest among the k random walks is returned, the incumbent is updated, and a new vertex $s \in S$ is explored. At the end, after all vertices in S have been visited, the incumbent contain the longest path found by the heuristic. We observe that the heuristic is not efficient for small instances because, as the parameter k grows, many repeated paths are generated starting at the same vertex.

Bökler *et al.* [5] proposed new integer linear programming formulations for the longest induced path problem, based on cut or subtour elimination constraints, exact separation routines, relaxation of variables and clique constraints. They obtain strictly stronger relaxations than those proposed in [13]. The core of their formulations is based on an extended graph $G^* = (V^*, E^*)$ by adding to G a new (universal) vertex s adjacent to all vertices of V, with $V^* = V \cup \{s\}$ and $E^* = E \cup \{(s, v) : v \in V\}$. It seeks for a longest induced cycle through s in G^*, ignoring induced chords incident to s. Searching for a cycle instead of a path, allows to require that each edge in the solution has exactly two adjacent edges that are also selected. Let $\delta^*(e) \subset E^*$ denote the edges adjacent to edge e in G^*. The partial formulation ILP_{Base} makes use of the binary variable y_e associated to each edge $e \in E^*$ of the graph:

$$y_e = y_{i,j} = \begin{cases} 1, & \text{if and only if edge } e = (i, j) \text{ is selected,} \\ 0, & \text{otherwise.} \end{cases}$$

$$\text{ILP}_{\text{Base}}: \qquad \max \sum_{e \in E} y_e \tag{2.10}$$

subject to:

$$\sum_{v \in V} y_{s,v} = 2, \tag{2.11}$$

$$2y_e \leq \sum_{f \in \delta^*(e)} y_f \leq 2, \qquad \forall e \in E, \tag{2.12}$$

$$y_e \in \{0, 1\}, \qquad \forall e \in E^*. \tag{2.13}$$

The objective function (2.10) maximizes the number of edges in the resulting path. Constraint (2.11) enforces that exactly two edges incident to vertex s are selected. To prevent the existence of chords, constraints (2.12) guarantee that any original edge $e \in E$ is adjacent to at most two selected edges. Furthermore, if edge $e \in E$ is selected, then precisely two of its adjacent edges must to be selected as well. Finally, constraints (2.13) correspond to the integrality requirements.

However, this formulation is not enough, because it still allows solutions formed by multiple disjoint cycles, only one of them containing vertex s. In order to obtain a longest single cycle through s yielding the longest induced path, one has to enforce that the graph induced by the optimal y-variables be connected. Bökler *et al.* [5] explored two strategies to achieve connectivity, by augmenting ILP_{Base} with cut or (generalized) subtour elimination constraints. Let $\delta^*(W) = \{(w, \bar{w}) \in E^* : w \in W, \bar{w} \in V^* \setminus W\}$ be the set of edges in the cut induced by $W \subseteq V^*$. The cut constraints can be formulated as:

$$\sum_{e \in \delta^*(v)} y_e \leq \sum_{e \in \delta^*(W)} y_e \qquad \forall W \subseteq V, \ \forall v \in W. \tag{2.14}$$

They guarantee that if a vertex v is incident to a selected edge, then any cut separating v from s contains at least two selected edges. Thus, there are (at least) two edge-disjoint paths selected between v and s. Considering the cycle properties of ILP_{Base}, we may conclude that all selected edges form a common cycle through s. The enhanced formulation obtained by adding the cut constraints is named ILP_{Cut}.

The above formulations can be further strengthened by introducing a set of additional clique inequalities. This, because if the subgraph $G[Q] = (Q, E[Q])$ induced in G by some vertex subset $Q \subseteq V$ is a clique, then any induced path may contain at most one edge of $E[Q]$:

$$\sum_{e \in E[Q]} y_e \leq 1, \qquad \forall Q \subseteq V : G[Q] \text{ is a clique.} \tag{2.15}$$

3. Exact enumerative algorithm and a new heuristic

In this section, we propose an exact algorithm and a new heuristic, both of them based on the enumeration of all induced paths of the graph, exploring an approach similar to that in [13].

3.1. Exact enumerative algorithm

Algorithm 1 explores all vertices of the graph $G = (V, E)$ to be considered as the source vertex s of an induced path. A recursive algorithm with backtracking is used to generate and explore all induced paths emanating from s. Procedure EXACT-ENUM takes as input the graph G. In lines 1 and 2, the longest induced path and the current path are initialized, respectively. The loop in lines 3–5 explores all vertices of the graph. For each vertex $s \in V$, line 4 makes a call to the recursive procedure INDUCED-PATHS-FROM-VERTEX outlined in Algorithm 2 to compute all induced paths in G emanating from the source s. Finally, line 6 returns the first longest induced path found during the exhaustive search.

Algorithm 1. EXACT-ENUM(G).

```
1: $P_{\max} \leftarrow \emptyset$
2: $P_{\text{temp}} \leftarrow \emptyset$
3: **for** each $s \in V$ **do**
4:     INDUCED-PATHS-FROM-VERTEX($G, s, P_{\max}, P_{\text{temp}}$)
5: **end for**
6: **return** $P_{\max}$
```

Algorithm 2. INDUCED-PATHS-FROM-VERTEX($G, s, P_{\max}, P_{\text{temp}}$).

```
1: $P_{\text{temp}} \leftarrow P_{\text{temp}} \oplus \langle s \rangle$
2: $N(s) \leftarrow \{t : (s,t) \in E, t \in V\}$
3: **if** $N(s) \neq \emptyset$ **then**
4:     **for** each $t \in N(s)$ **do**
5:         $V_{\text{temp}} \leftarrow V \setminus (\{s\} \cup N(s) \setminus \{t\})$
6:         $E_{\text{temp}} \leftarrow \{(i,j) \in E : i \in V_{\text{temp}}, j \in V_{\text{temp}}\}$
7:         $G_{\text{temp}} \leftarrow (V_{\text{temp}}, E_{\text{temp}})$
8:         INDUCED-PATHS-FROM-VERTEX($G_{\text{temp}}, t, P_{\max}, P_{\text{temp}}$)
9:     **end for**
10: **else**
11:     **if** $|P_{\text{temp}}| > |P_{\max}|$ **then**
12:         $P_{\max} \leftarrow P_{\text{temp}}$
13:     **end if**
14: **end if**
15: $P_{\text{temp}} \leftarrow P_{\text{temp}} \ominus \langle s \rangle$
```

Procedure INDUCED-PATHS-FROM-VERTEX takes as inputs the graph G, the source vertex s, the longest found induced path $P_{\max}$ and the current path P_{temp}. In line 1, the input vertex s is appended to the end of the current path P_{temp} under construction. Next, we obtain in line 2 the set of vertices $N(s)$ adjacent to vertex s. If $N(s)$ is not empty, then the loop in lines 4–9 visits every vertex t adjacent to s. For each vertex t, lines 5–7 create the subgraph $G_{\text{temp}} = (V_{\text{temp}}, E_{\text{temp}})$ induced in G by $V \setminus (\{s\} \cup N(s) \setminus \{t\})$, since the vertices in $N(s) \setminus \{t\}$ can not belong to any induced path containing s and t. Vertex eliminations can be considered as cuts in the search space, because they discard the examination of paths that can not be induced paths. Finally, line 8 invokes recursively the procedure INDUCED-PATHS-FROM-VERTEX in the new induced graph G_{temp} from the new source vertex t that will be appended to the new, current path P_{temp} obtained in line 1. Otherwise, in case vertex s has no neighbors, then it is a leaf of the search tree and the current path P_{temp} can not be further extended. Then, line 11 checks if the size of path P_{temp} is greater than that of path $P_{\max}$. If this is true, then we update $P_{\max}$ in line 12. At the end of procedure INDUCED-PATHS-FROM-VERTEX, in line 15, we remove from path P_{temp} the vertex s previously added in line 1.

3.2. Heuristic

The new heuristic also explores all vertices of the graph $G = (V, E)$ as possible source vertices of induced paths. Procedure HLIPP whose pseudo-code appears in Algorithm 3 is very similar to Algorithm 1. However, it also takes as input an additional parameter *maxpaths* that limits the number of induced paths that are explored from each source vertex s. In lines 1 and 2, the longest induced path and the current path are initialized, respectively. The loop in lines 3–8 explores all vertices of the graph. Variable *#paths* is reset to 0 in line 4 at each iteration of the loop to count the number of induced paths explored from each source vertex $s \in V$. Variable *last_improv* is reset to 0 in line 5 before vertex s is explored. Variable *truncated* is set to `False` in line 6. Line 7 performs the call to the recursive procedure FIRST-INDUCED-PATHS. Line 9 returns the longest induced path $P_{\max}$ found.

Algorithm 3. HLIPP($G, maxpaths$).

```
1: $P_{\max} \leftarrow \emptyset$
2: $P_{\text{temp}} \leftarrow \emptyset$
3: for each $s \in V$ do
4:     $\#paths \leftarrow 0$
5:     $last\_improv \leftarrow 0$
6:     $truncated \leftarrow$ False
7:     FIRST-INDUCED-PATHS($G, s, P_{\max}, P_{\text{temp}}, maxpaths, \#paths, last\_improv, truncated$)
8: end for
9: return $P_{\max}$
```

Procedure FIRST-INDUCED-PATHS outlined in Algorithm 4 takes as inputs the graph G, the source vertex s, the longest found induced path $P_{\max}$, the current path P_{temp}, the parameter *maxpaths* that is used as the stopping criterion to interrupt the search of the current vertex s, the counter *#paths* of maximal induced paths generated from vertex s, the parameter *last_improv* that keeps track of the last time an improving solution was found, and a flag *truncated* that is used to indicate that the stopping criterion was reached for the vertex being explored. In line 1 we append the input vertex s to the end of the current path P_{temp} under construction. Next, we obtain in line 2 the set of vertices $N(s)$ adjacent to vertex s. If $N(s)$ is not empty, then the loop in lines 4–12 visits every vertex t adjacent to s, unless a stopping criterion is met. For each vertex t, lines 5–7 create de subgraph $G_{\text{temp}} = (V_{\text{temp}}, E_{\text{temp}})$ induced in G by $V \setminus (\{s\} \cup N(s) \setminus \{t\})$, since the vertices in $N(s) \setminus \{t\}$ can not belong to any induced path containing s and t. Next, line 8 invokes recursively the procedure FIRST-INDUCED-PATHS in the new induced graph G_{temp}, from the new source vertex t that will be appended to the new, current path P_{temp} obtained in line 1. If the stopping criterion for the exploration of vertex s is reached during this call, then the value of the flag *truncated* is returned as `True` and propagated backwards in lines 9–11. Otherwise, in case vertex s has no neighbors, then it is a leaf of the search tree and the current path P_{temp} can not be further extended. Then, line 14 increases by one the number of maximal induced path generated. Next, line 15 checks if the size of path P_{temp} is greater than the size of path $P_{\max}$. If this is true, then $P_{\max}$ is updated in line 16 and the variable *last_improv*, which accounts for the last time an improving solution was found, is updated in line 17. Line 19 implements the stopping criterion for the vertex under exploration. If it determines that more than *maxpaths* paths have been generated since the last improvement of the incumbent, then the search will be truncated in line 22, after emptying out P_{temp} in line 20 and resetting the flag *truncated* to `True` in line 21. Finally, in line 25 at the end of procedure, vertex s previously added in line 1 is removed from path P_{temp}.

Algorithm 4. FIRST-INDUCED-PATHS($G, s, P_{\max}, P_{\text{temp}}, maxpaths, \#paths, last\_improv, truncated$).

```
 1: $P_{\text{temp}} \leftarrow P_{\text{temp}} \oplus \langle s \rangle$
 2: $N(s) \leftarrow \{t : (s,t) \in E, t \in V\}$
 3: if $N(s) \neq \emptyset$ then
 4:     for each $t \in N(s)$ do
 5:         $V_{\text{temp}} \leftarrow V \setminus (\{s\} \cup N(s) \setminus \{t\})$
 6:         $E_{\text{temp}} \leftarrow \{(i,j) \in E : i \in V_{\text{temp}}, j \in V_{\text{temp}}\}$
 7:         $G_{\text{temp}} \leftarrow (V_{\text{temp}}, E_{\text{temp}})$
 8:         FIRST-INDUCED-PATHS($G_{\text{temp}}, t, P_{\max}, P_{\text{temp}}, maxpaths, \#paths, last\_improv, truncated$)
 9:         if $truncated = $ True then
10:             return
11:         end if
12:     end for
13: else
14:     $\#paths \leftarrow \#paths + 1$
15:     if $|P_{\text{temp}}| > |P_{\max}|$ then
16:         $P_{\max} \leftarrow P_{\text{temp}}$
17:         $last\_improv \leftarrow \#paths$
18:     end if
19:     if $\#paths - last\_improv > maxpaths$ then
20:         $P_{\text{temp}} \leftarrow \emptyset$
21:         $truncated \leftarrow$ True
22:         return
23:     end if
24: end if
25: $P_{\text{temp}} \leftarrow P_{\text{temp}} \ominus \langle s \rangle$
```

4. Computational experiments

In this section, we present computational experiments with the exact algorithm and the new heuristic proposed in Section 3 and the algorithms of Matsypura *et al.* [13] and Bökler *et al.* [5].

4.1. Test instances

Matsypura *et al.* [13] considered two sets of real-life graphs used in their computational experiments. The first set consists of network instances that are commonly used in the literature. They sought the longest induced path in the largest connected component of each instance. The second set is formed by networks representing the social connectivity of character interactions in different films and series [14]. They also generated synthetic instances with 20, 30 and 40 vertices and different edge densities, using the Barabsi-Albert graph generating model [3].

Bökler *et al.* [5] considered new instances, in addition to those proposed by Matsypura *et al.* [13]. Their instances are grouped into four sets: RWC, MG, BAS and BAL. The first set (RWC) is a collection of 22

TABLE 1. CPU performance comparison, data extracted from [17]: Higher values represent better performance. Second to last columns present the hardware used in this paper, in [13], and in [5], respectively.

Benchmarks	Intel Core i5-4460S	Intel Xeon E5-1650 v2	Intel Xeon Gold 6134
Clock speed (GHz)	2.9	3.5	3.2
Turbo speed (GHz)	Up to 3.4	Up to 3.9	Up to 3.7
CPU single thread rating	1822	1929	2247
CPU mark rating	4383	8977	16 687

real-world networks, including communication and social networks of companies and of characters in books, as well as transportation, biological, and technical networks. The second set (MG) consists of 773 graphs representing social networks of movie characters, of which Matsypura *et al.* [13] considered only 17 of them. The instances in the other two sets were generated with the Barabsi-Albert probabilistic model for scale-free networks. The third set (BAS) has four groups of 30 graphs created with the same parameter values considered by Matsypura *et al.* [13], each group with $(|V|, d) \in \{(20, 3), (30, 3), (40, 3), (40, 2)\}$, where $d = (|E| + 1)/|V|$ as defined by the authors. They also considered a fourth set (BAL) of graphs with 100 vertices and for each value of $d \in \{2, 3, 10, 30, 50\}$ they generated 30 instances.

4.2. Computational environment

We implemented the exact enumerative algorithm and the new heuristic proposed in Section 3 in C++ with the GNU GCC compiler version 5.4.0. All experiments on these algorithms were performed on a computer with a four-processor 2.90 GHz Intel Core i5-4460S CPU with 8 GB of RAM running the operating system Linux Ubuntu 16.04 LTS of 64 bits.

All experiments executed in [13] were conducted on a MacPro (late 2013) with a 3.5 GHz 6-Core Intel Xeon E5 and 32 GB of RAM running macOS the operating system High Sierra using a single thread. Their models were implemented in Anaconda Python 2.7 and Gurobi 8.1 was used as the IP solver. All Gurobi parameters were set to their default values, with each run limited to one single thread and a time limit of eight hours.

Bökler *et al.* [5] run all their computational tests on an Intel Xeon Gold 6134 machine with 3.2 GHz and 256 GB of RAM running the operating system Debian 9. The authors used C++ (GCC 8.3.0) and limited each run to one single thread with a time limit of 20 min and a memory limit of 8 GB of RAM.

Table 1 based on the benchmarks in [17] addresses the performance of the computers involved in the three experiments. Although the computers used by Matsypura *et al.* [13] and Bökler *et al.* [5] are faster than ours, we will compare the execution times of the algorithms on their own, independently of the machines on where the experiments had been executed. In other words, we will not take into account the differences in performance of the three different processors. For a more detailed evaluation, the interested reader may use the information in this table.

4.3. Exact algorithms

In this section, we compare our exact enumerative algorithm with the state-of-the-art integer programming approaches proposed by Matsypura *et al.* [13] and Bökler *et al.* [5], with a time limit of 20 min and one single thread for each run.

Bökler *et al.* [5] denoted by "W" their implementation of the exact algorithm in [13]. The authors considered various parameter settings in their implementation of model ILP$_{\text{Cut}}$, with all resulting algorithms denoted by a "C" with subscripts and superscripts defining the parameters: subscript "frac" denotes the use of fractional separation in addition to integral separation; superscript "n" shows that vertex variables are the unique integer

Table 2. Running times in seconds of the exact algorithms on the RWC instances.

Instance	OPT	$\|V\|$	$\|E\|$	EXACT-ENUM	W	Tmin	Cmin	Tmax	Cmax
`high-tech`	13	33	91	**0.15**	15.40	0.41	$C^{n,c}_{int}$	3.15	C^{c}_{frac}
`karate`	9	34	78	**0.04**	2.98	0.66	$C^{n,c}_{int}$	3.71	C^{n}_{frac}
`mexican`	16	35	117	**0.81**	73.30	0.87	$C^{n,c}_{int}$	3.59	C^{c}_{frac}
`sawmill`	18	36	62	**0.06**	70.00	0.43	C_{frac}	3.34	$C^{n,c}_{frac}$
`tailorS1`	13	39	158	**1.05**	83.80	1.51	C^{n}_{int}	7.92	C_{frac}
`chesapeake`	16	39	170	**0.78**	106.00	1.84	C_{int}	13.11	C_{frac}
`tailorS2`	15	39	223	**2.13**	445.00	2.89	$C^{n,c}_{int}$	21.78	C_{frac}
`attiro`	31	59	128	71.35	◷	**0.89**	$C^{n,c}_{int}$	2.57	C_{frac}
`krebs`	17	62	153	**0.80**	522.00	2.33	$C^{n,c}_{frac}$	28.21	C_{frac}
`dolphins`	24	62	159	91.77	◷	**2.99**	C^{n}_{frac}	27.59	C_{frac}
`prison`	36	67	142	64.15	◷	**1.02**	$C^{n,c}_{int}$	13.36	C_{int}
`huck`	9	69	297	**0.30**	41.70	5.96	$C^{n,c}_{int}$	◷	C_{int}
`sanjuansur`	38	75	144	645.81	◷	**3.65**	C^{n}_{frac}	30.67	C_{int}
`jean`	11	77	254	**1.18**	121.00	3.88	$C^{n,c}_{int}$	464.89	C_{int}
`david`	19	87	406	23.35	◷	**6.93**	$C^{n,c}_{int}$	719.46	C_{frac}
`ieeebus`	47	118	179	322.40	◷	**3.13**	C^{n}_{frac}	39.82	C^{c}_{int}
`sfi`	13	118	200	**0.32**	44.40	2.44	$C^{n,c}_{frac}$	47.41	C_{int}
`anna`	20	138	493	17.17	◷	**7.09**	$C^{n,c}_{int}$	439.23	C^{n}_{int}
`usair`	46	332	2126	◷	◷	**922.94**	$C^{n,c}_{int}$	◷	all but $C^{n,c}_{int}$
`494bus`	142	494	586	◷	◷	**170.74**	$C^{n,c}_{frac}$	◷	$C_{int}, C^{c}_{int}, C^{n}_{int}, C^{n,c}_{int}$

variables. The superscript "c" indicates the use of clique constraints. The authors considered thereby all possible combinations of parameters of ILP_{Cut}, corresponding to eight different implementations.

Table 2 displays the computational experiments on the RWC instances (as named by Bökler *et al.* [5]). The second column gives the optimal value. The third and fourth columns show the number of vertices and edges in each instance, respectively. The fifth column displays the time in seconds taken by our exact enumerative algorithm. The sixth column shows the time in seconds obtained by the exact algorithm of [13], as implemented in [5]. The next two columns, indicated respectively by Tmin and Cmin, present the minimum running time over all eight implementations of ILP_{Cut} and the fastest variant. The last two columns, indicated respectively by Tmax and Cmax, present the maximum running time over all eight implementations of ILP_{Cut} and the slowest variant (or those that timed out). Timeouts are denoted by ◷ and the minimum times were marked in boldface. Instances `yeast` and `622bus` (not presented in the table) could not be solved by any of the algorithms within the maximum time limit.

The table shows that, for the RWC instances, the new enumerative algorithm performed better in terms of running times and the number of instances solved to optimality than the exact algorithm in [13]. However, the comparison with respect to the implementations of ILP_{Cut} is less straightforward. It is noticeable to observe that the fastest and the slowest variants of ILP_{Cut} change from instance to instance. Variant $C^{n,c}_{int}$ was the

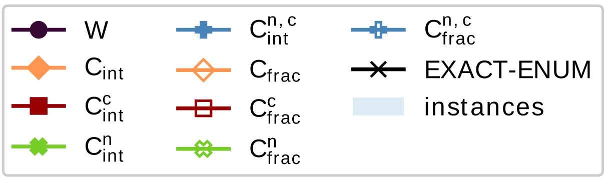

Figure 1. Legend with the identifications of the algorithms and ILP_{Cut} implementations.

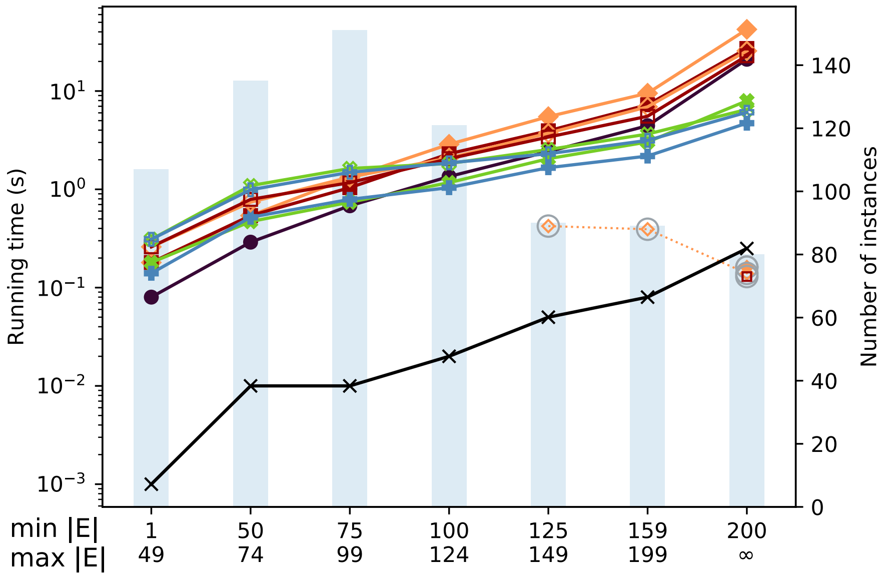

Figure 2. Running times in seconds of the exact algorithms and the ILP implementations on the MG instances.

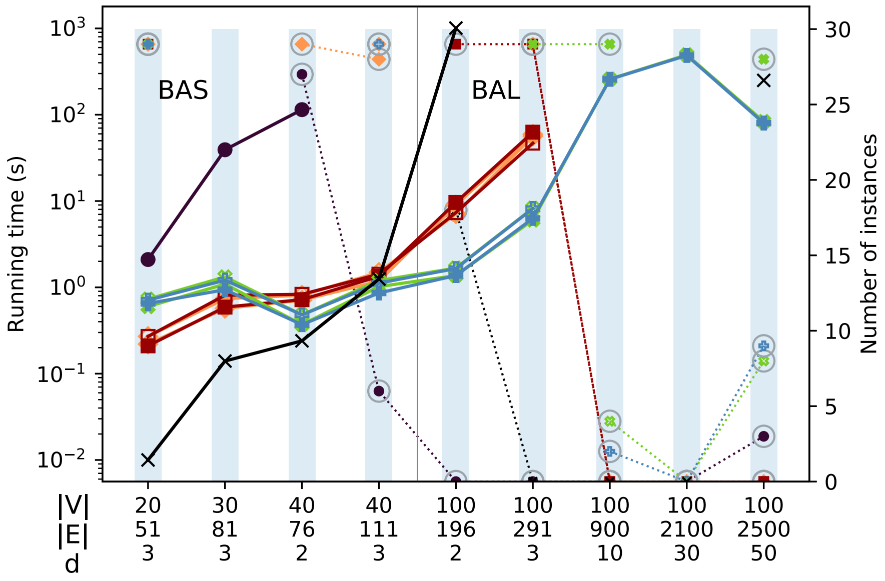

Figure 3. Running times in seconds of the exact algorithms and the ILP implementations on the BAS and BAL instances.

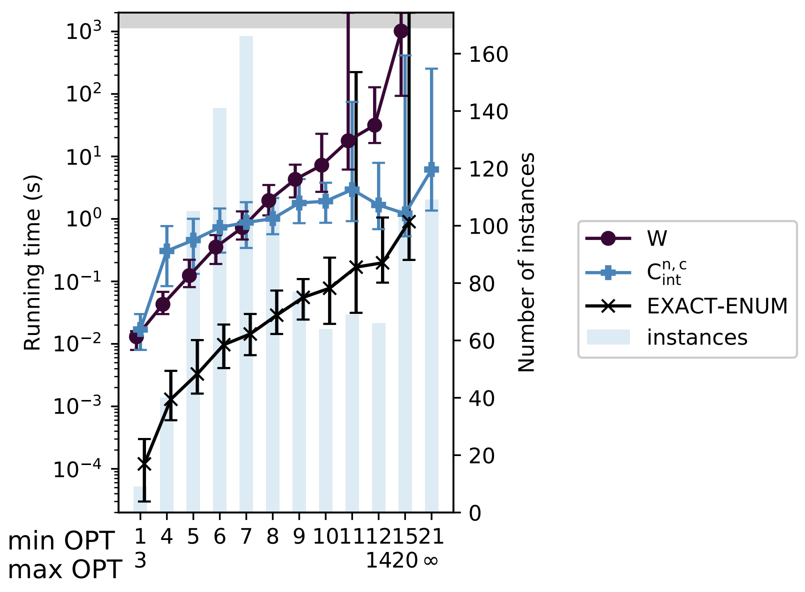

FIGURE 4. Running times *vs.* optimal values for all instances: whiskers mark the 20% and 80% percentiles. The gray area on top of the plot marks timeouts. Each run was limited to 20 min.

fastest for 11 out of 22 instances and the only to solve `usair`, but did not solve instance `494bus` within the time limit. Variants C_{int}^{c} and C_{frac}^{c} never were the fastest. The new enumerative algorithm was faster than the fastest variant of $\mathrm{ILP}_{\mathrm{Cut}}$ for 11 out of 22 instances. For two instances, the new algorithm was slower than the fastest, but faster than the slowest variant of $\mathrm{ILP}_{\mathrm{Cut}}$. The new enumerative algorithm was slower than the worst variant of $\mathrm{ILP}_{\mathrm{Cut}}$ for five out of the 22 RWC instances. Still, for two instances the new enumerative algorithm did not find the optimal solution, but the fastest variant of $\mathrm{ILP}_{\mathrm{Cut}}$ did. Overall, we may say that the new enumerative algorithm is competitive with the integer programming formulations $\mathrm{ILP}_{\mathrm{Cut}}$ in [5], in particular if one considers the differences in performance of the processors involved in the computational experiments and the fact that the best variant of $\mathrm{ILP}_{\mathrm{Cut}}$ for each instance can not be predicted beforehand and choosing the best among the eight $\mathrm{ILP}_{\mathrm{Cut}}$ implementations is way far from being a clear decision.

Figure 1 shows the legend with the identification of the algorithms and the $\mathrm{ILP}_{\mathrm{Cut}}$ implementations whose results are displayed in Figures 2 and 3. Figure 2 displays comparative results for the MG instances, with the horizontal axis indicating the minimum and the maximum number of edges in the instances in each category (same abscissa). Figure 3 shows the results for the instances in sets BAS and BAL, with the horizontal axis indicating the number of vertices, the number of edges, and the parameter d for the instances in each category (same abscissa). Vertical bars in light blue in the background give the number of instances in each category. For each exact algorithm (W, the eight variants of $\mathrm{ILP}_{\mathrm{Cut}}$, and the new enumerative algorithm), we represent the median of the running times over all instances in the same category. Points represented by gray encircled markers connected by dotted lines show the number of solved instances (when not all in the same category have been solved to optimality).

We observe in Figure 2 that the new exact enumerative algorithm solved all 773 MG instances using the smallest running times, with respect to all other algorithms. Figure 3 shows that the new exact enumerative algorithm remains competitive and globally the fastest on average also for the BAS instances. However, both the enumerative algorithm and that in [13] do not perform well for the BAL instances, where the latter failed for virtually all instances, while some implementations of $\mathrm{ILP}_{\mathrm{Cut}}$ were faster and solved more instances than the new exact enumerative algorithm.

Table 3. Comparative numerical results between the randomized heuristic [13] and heuristic HLIPP on large graphs.

Name	$\|V\|$	$\|E\|$	OPT	Randomized heuristic						HLIPP					
				H100		H1000		H10000		LH100		LH1000		LH10000	
				Time (s)	Obj.	Time (s)	Obj.	Time (s)	Obj.	Time (s)	Obj.	Time (s)	Obj.	Time (s)	Obj.
`abyss`	28	108	6	0.21	**6**	2.1	**6**	21	**6**	0.01	**6**	0.01	**6**	0.01	**6**
`hi-tech`	33	91	13	0.39	**13**	3.8	**13**	37	**13**	0.03	**13**	0.15	**13**	0.15	**13**
`karate`	34	78	9	0.31	**9**	2.9	**9**	30	**9**	0.02	**9**	0.04	**9**	0.04	**9**
`mexican`	35	117	16	0.53	15	5.3	15	53	**16**	0.04	**16**	0.36	**16**	0.76	**16**
`sawmill`	36	62	18	0.41	**18**	4.1	**18**	41	**18**	0.04	**18**	0.06	**18**	0.05	**18**
`pulp fiction`	38	102	7	0.27	**7**	2.6	**7**	26	**7**	0.02	**7**	0.02	**7**	0.02	**7**
`chesapeake`	39	170	16	0.56	**16**	5.5	**16**	55	**16**	0.05	**16**	0.42	**16**	0.73	**16**
`tailorS1`	39	158	13	0.58	12	5.7	**13**	58	**13**	0.05	**13**	0.38	**13**	1.00	**13**
`tailorS2`	39	223	15	0.64	13	6.4	14	64	**15**	0.06	12	0.49	13	1.97	**15**
`romeo and juliet`	41	120	9	0.35	8	3.5	**9**	35	**9**	0.03	**9**	0.07	**9**	0.06	**9**
`oceans 12`	42	147	8	0.40	**8**	3.9	**8**	39	**8**	0.04	**8**	0.07	**8**	0.07	**8**
`die hard`	47	237	10	0.50	9	5.0	**10**	49	**10**	0.05	**10**	0.11	**10**	0.11	**10**
`star wars II`	47	148	8	0.41	**8**	4.1	**8**	41	**8**	0.04	**8**	0.06	**8**	0.05	**8**
`oceans 11`	50	145	9	0.51	**9**	5.0	**9**	48	**9**	0.05	8	0.08	**9**	0.07	**9**
`the departed`	51	117	8	0.45	**8**	4.5	**8**	45	**8**	0.04	**8**	0.05	**8**	0.05	**8**
`krebs`	62	153	17	0.78	13	7.7	16	77	**17**	0.09	15	0.59	**17**	0.75	**17**
`philadelphia`	65	258	10	0.68	8	6.8	9	67	9	0.09	9	0.31	**10**	0.30	**10**
`2012`	66	211	12	0.86	10	8.4	11	84	**12**	0.09	11	0.62	**12**	0.70	**12**
`braveheart`	67	310	11	0.82	9	8.1	9	81	10	0.09	**11**	0.60	**11**	0.64	**11**
`huck`	69	297	9	0.75	8	7.4	**9**	74	**9**	0.11	**9**	0.30	**9**	0.28	**9**
`gandhi`	76	200	10	0.89	9	8.8	**10**	88	**10**	0.11	9	0.16	**10**	0.15	**10**
`watchmen`	76	201	9	0.79	**9**	7.8	**9**	80	**9**	0.10	**9**	0.26	**9**	0.24	**9**
`jean`	77	254	11	0.97	10	9.6	**11**	96	**11**	0.12	**11**	0.80	**11**	1.10	**11**
`godfather II`	78	219	18	1.03	13	10.2	14	103	15	0.12	16	0.90	**18**	1.23	**18**
`catch me if you can`	82	162	8	0.94	**8**	9.3	**8**	93	**8**	0.11	**8**	0.17	**8**	0.15	**8**
`david`	87	406	19	1.48	14	14.8	15	149	17	0.23	18	1.87	**19**	14.40	**19**
`doors`	95	567	12	1.65	9	16.3	11	163	**12**	0.30	**12**	2.17	**12**	2.75	**12**
`public enemies`	99	317	20	1.55	12	15.5	15	154	15	0.20	17	1.57	**20**	3.58	**20**
`santafe`	118	200	13	1.39	9	13.9	10	138	10	0.19	**13**	0.30	**13**	0.29	**13**
`anna`	138	493	20	3.02	13	30.0	15	303	16	0.42	16	3.36	**20**	16.01	**20**
`attiro`	59	128	31	1.53	27	15	28	142	29	0.11	28	0.92	29	8.77	30
`dolphins`	62	159	24	1.38	21	14	22	137	**24**	0.11	20	0.98	22	9.11	23
`prison`	67	142	36	1.52	27	15	30	152	32	0.13	28	1.23	31	11.66	**36**
`sanjuansur`	75	144	38	2.09	30	21	33	213	36	0.16	35	1.42	35	14.26	37
`ieeebus`	118	179	47	3.29	29	33	31	329	38	0.37	44	3.20	44	30.95	**47**
`USAir97`	332	2126	46	21.07	22	206	27	2044	30	3.50	29	37.88	33	347.12	38
`494_bus`	494	586	142	41.56	45	413	49	3901	61	7.99	101	70.25	107	738.86	114
`662_bus`	662	906	?	141.00	90	1402	92	13 819	110	26.33	242	172.08	242	1608.83	242
`S.Cerevisae`	1458	1948	?	290.62	35	2855	38	28 221	43	76.60	144	451.74	149	4139.20	155
# of optimal values			37		12		18		24		20		29		32

Bökler *et al.* [5] observed that the running time of the exact algorithm in [13] heavily depends on the optimal value of each instance, while their new implementations (*e.g.* $C_{\text{int}}^{n,c}$) are less dependent on the solution size. This is illustrated in Figure 4, which correlates the median running times of three exact algorithms with the size of the longest induced path (OPT), considering all the test instances solved by at least one of the exact methods. The horizontal axis of the figure indicates the minimum and the maximum optimal values for the instances represented in the same category with the same abscissa. The running time of the new exact enumerative algorithm is naturally expected to depend on the optimal value, as illustrated by the figure.

4.4. Heuristics

In this set of experiments, we compare the performance of the HLIPP heuristic described in Algorithm 3 with the randomized heuristic of Matsypura *et al.* [13] on the same instances considered by its authors.

TABLE 4. Comparative numerical results between the exact algorithm EXACT-ENUM and the heuristic HLIPP (LH10000 corresponds to Algorithm 3 with 10 000 as the stopping criterion).

	EXACT-ENUM		HLIPP (LH10000)	
Name	OPT	Time (s)	Time (s)	Obj.
`abyss`	6	0.01	0.01	**6**
`hi-tech`	13	0.15	0.15	**13**
`karate`	9	0.04	0.04	**9**
`mexican`	16	0.81	0.76	**16**
`sawmill`	18	0.06	0.05	**18**
`pulp fiction`	7	0.02	0.02	**7**
`chesapeake`	16	0.78	0.73	**16**
`tailorS1`	13	1.05	1.00	**13**
`tailorS2`	15	2.13	1.97	**15**
`romeo and juliet`	9	0.06	0.06	**9**
`oceans 12`	8	0.07	0.07	**8**
`die hard`	10	0.12	0.11	**10**
`star wars II`	8	0.06	0.05	**8**
`oceans 11`	9	0.07	0.07	**9**
`the departed`	8	0.05	0.05	**8**
`krebs`	17	0.80	0.75	**17**
`philadelphia`	10	0.31	0.30	**10**
`2012`	12	0.71	0.70	**12**
`braveheart`	11	0.66	0.64	**11**
`huck`	9	0.30	0.28	**9**
`gandhi`	10	0.16	0.15	**10**
`watchmen`	9	0.25	0.24	**9**
`jean`	11	1.18	1.10	**11**
`godfather II`	18	1.26	1.23	**18**
`catch me if you can`	8	0.17	0.15	**8**
`david`	19	23.35	14.40	**19**
`doors`	12	2.77	2.75	**12**
`public enemies`	20	3.60	3.58	**20**
`santafe`	13	0.32	0.29	**13**
`anna`	20	17.17	16.01	**20**
`attiro`	31	71.35	8.77	30
`dolphins`	24	91.77	9.11	23
`prison`	36	64.15	11.66	**36**
`sanjuansur`	38	645.81	14.26	37
`ieeebus`	47	322.40	30.95	**47**
`USAir97`	46	🕒	347.12	38
`494_bus`	142	🕒	738.86	114
`662_bus`	?	🕒	1608.83	242
`S.Cerevisae`	?	🕒	4139.20	155
# of optimal values	37			32

Comparative computational results are presented in Table 3. The four first columns present the name of the instance, the number of vertices, the number of edges, and the size of the longest induced path (OPT) for each graph (unknown for the two last instances). Next, each pair of columns give the time in seconds and the best solution value obtained by one run of the randomized heuristic [13] with 100, 1000 and 10 000 restarts (columns H100, H1000, and H10000, respectively). Columns LH100, LH1000 and LH10000 give the same information for

Algorithm 3, with 100, 1000 and 10 000 as the limit on the number of paths (*maxpaths*) that are explored from each source vertex, respectively.

The randomized heuristic stops after generating a given number of random induced paths (named "restarts" by the authors) starting from each vertex of some given subset $S \subseteq V$. Therefore, it implements diversification by randomization. The proposed HLIPP deterministic heuristic stops after generating a sequence of *maxpaths* induced paths from each vertex of the graph that do not improve the incumbent solution. Our approach explores more induced paths, leading to a potentially better solution. In addition, it generates the induced paths more efficiently, which makes HLIPP faster for all test instances in the computational experiments.

In fact, Table 3 shows that the running times of the proposed HLIPP heuristic are consistently smaller than those reported for the randomized heuristic in [13]. HLIPP is also more robust, since it found significantly more optimal values (or the best known solution values). The quality of heuristic HLIPP and the difference in performance of the two heuristics in terms of solution quality and running times are particularly clear for the two last instances with unknown optimal value in the bottom of the table.

Table 4 compares the exact algorithm EXACT-ENUM (Algorithm 1) with heuristic HLIPP (Algorithm 3) on all 39 instances in Table 3. Heuristic HLIPP (running with $maxpaths = 10\,000$) found the optimal value for 32 instances (out of 37 whose optimal values are known) in time not greater than that taken by our exact algorithm.

5. Analysis of the optimal solutions

In this section, we investigate the correlation of the running times and sizes of the longest induced paths with some characteristics of the solved instances of the four test sets (RWC, MG, BAS, BAL) described in Section 4.1. We also analyze the eccentricity and the degree of the extreme vertices of the optimal solutions obtained with our exact algorithm.

Figure 5 shows the Pearson correlation coefficient [18] between the running times of the instances solved by our exact enumeration algorithm and some graph characteristics: average vertex degree, average vertex eccentricity, average pairwise distance, global clustering coefficient, and average vertex (local) clustering coefficient. The global clustering coefficient is the ratio between three times the number of triangles and the number of pairs of adjacent edges in a graph [19]. This measure is also called the transitivity and represents the probability that two vertices that are adjacent to a third vertex will be adjacent themselves [16]. The (local) clustering coefficient

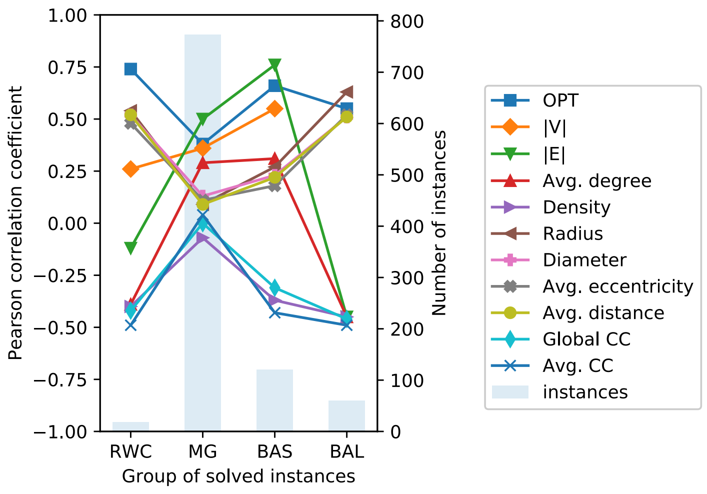

FIGURE 5. Pearson correlation coefficient [18] between the running times of the instances solved by our exact enumeration algorithm EXACT-ENUM and some graph characteristics.

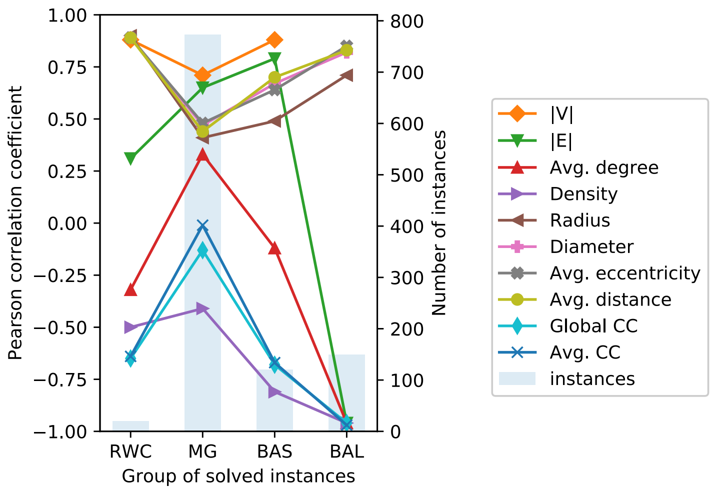

FIGURE 6. Pearson correlation coefficient between the optimal size OPT and some graph characteristics on the instances solved by at least one exact algorithm.

TABLE 5. Main quantitative information for each set of test instances exactly solved by algorithm EXACT-ENUM. Optimal solutions correspond to the number of distinct undirected longest induced paths. Distinct extremities account for the number of different extremities calculated over all optimal solutions.

	RWC	MG	BAS	BAL	Total
Solved instances	18	773	120	60	971
Optimal solutions	504	20 859	1025	59 292	81 680
Distinct extremities	119	5523	619	732	6993

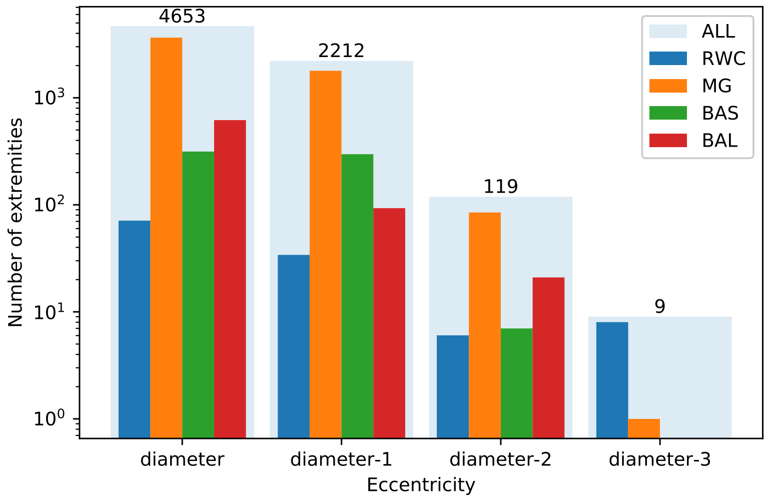

FIGURE 7. Number of extremities in the optimal paths *vs.* their eccentricities.

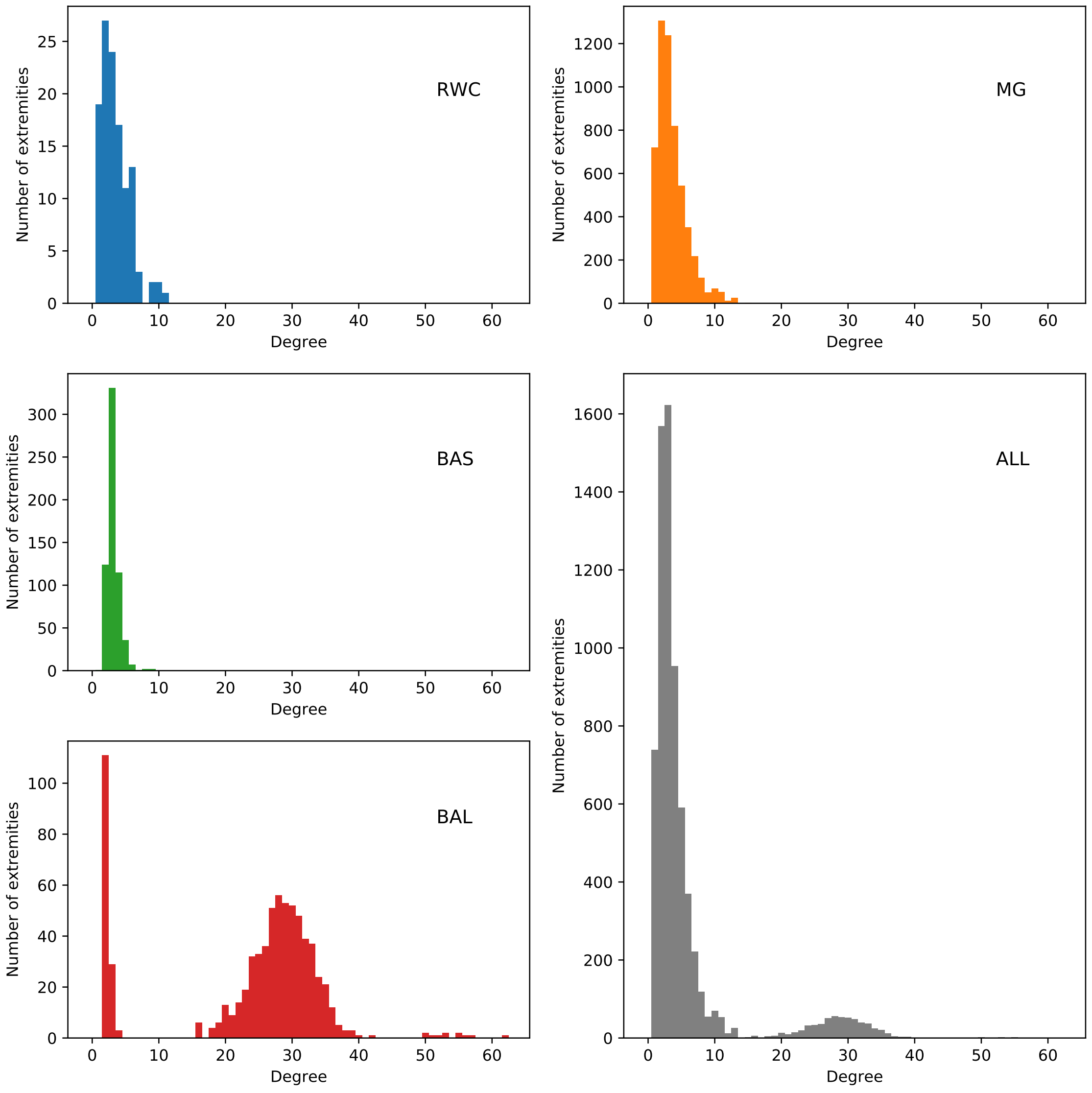

FIGURE 8. Number of extremities in the optimal paths *vs.* their degrees.

C measures the cliquishness of a typical neighbourhood and is defined as follows. Suppose that a vertex v has k_v neighbours. Then, at most $k_v(k_v - 1)/2$ edges can exist between them. Let C_v denote the fraction of these possible edges that actually exist. Define C as the average of C_v over all vertices v of the graph. For friendship networks, this statistic has an intuitive meaning: C_v reflects the extent to which friends of v are also friends of each other and, consequently, C measures the cliquishness of a typical friendship circle [20]. NetworkX [15] was used to compute all graph characteristics. Each set is represented as a category.

Figure 5 shows a high positive correlation between the running times and the size OPT of the longest induced paths (as already illustrated in Fig. 4), as well as between the running times and the number $|V|$ of vertices in the graph. Some characteristics, such as the number of edges $|E|$, are highly correlated with the running times for some sets of test instances but not for others, while others, such as the graph density, have a negative correlation with the running times for all test sets, with a value very close to zero for the MG instances.

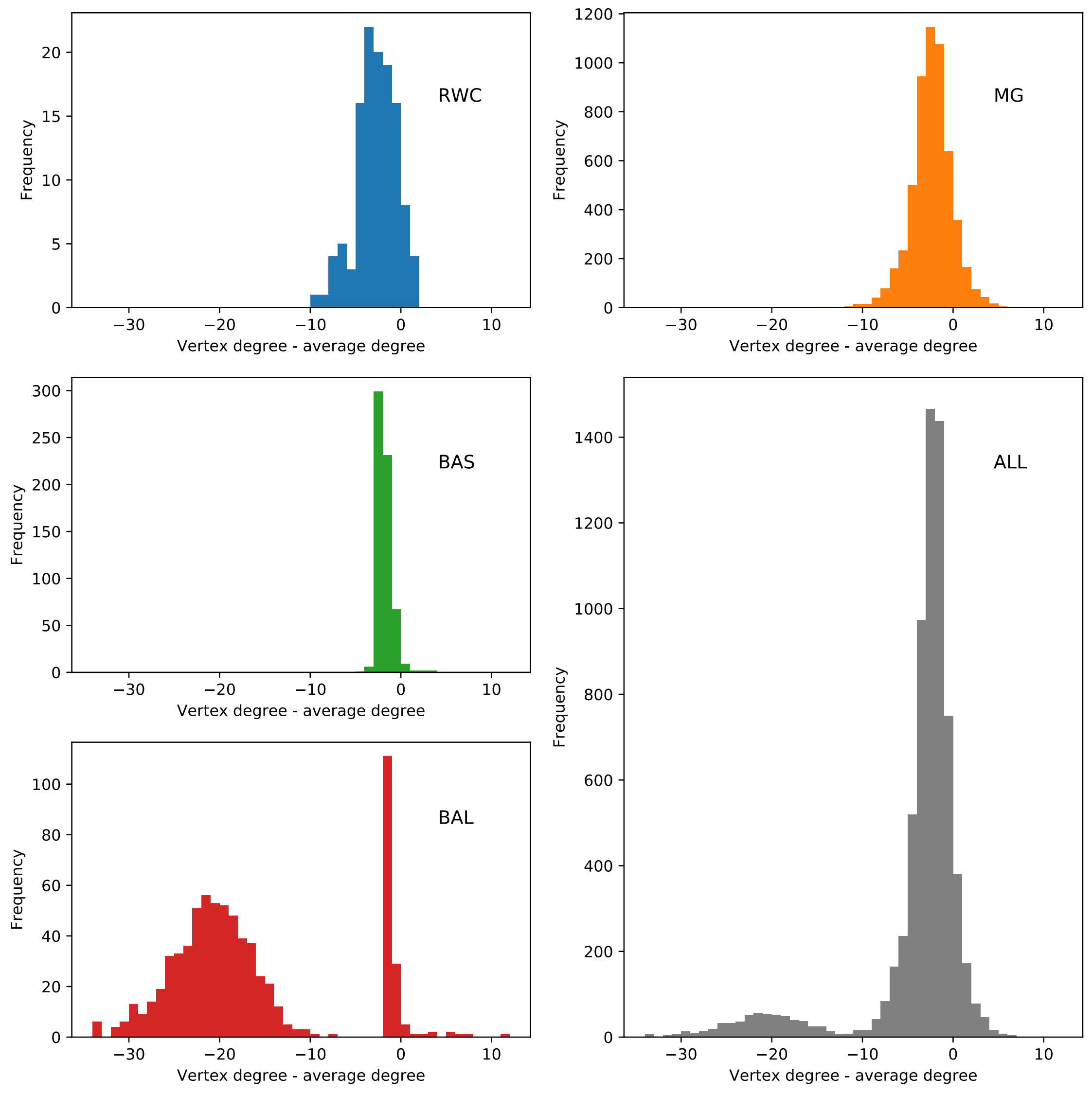

FIGURE 9. Frequency histogram of the degree differences of the extremities of the optimal solutions.

Figure 6 shows the Pearson correlation coefficient between the size OPT of the longest induced paths and some graph characteristics. It shows a high positive correlation between the size OPT of the longest induced paths and the number $|V|$ of vertices in the graph.

We also investigate the eccentricity and the degree of the extremities of the optimal paths obtained by our exact algorithm.

Table 5 summarizes the main quantitative information for each set of test instances exactly solved by algorithm EXACT-ENUM. We observe that although the 60 BAL instances represent only 6.2% of the total, they contribute with 72.6% of the optimal solutions found. On the other hand, the 773 MG instances represent 79.6% of the total, but contribute with only 25.5% of the optimal solutions.

Figure 7 relates the number of extremities of the optimal paths with a diameter-based measure of their eccentricity. We provide the global information (considering all test instances exactly solved to optimality by

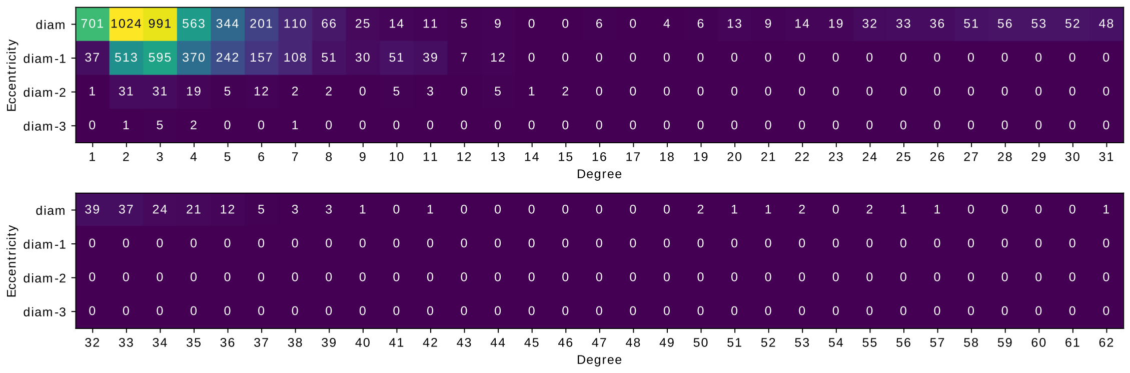

FIGURE 10. Number of extremities of optimal solutions with specific eccentricity and degree (over all instances solved by algorithm EXACT-ENUM).

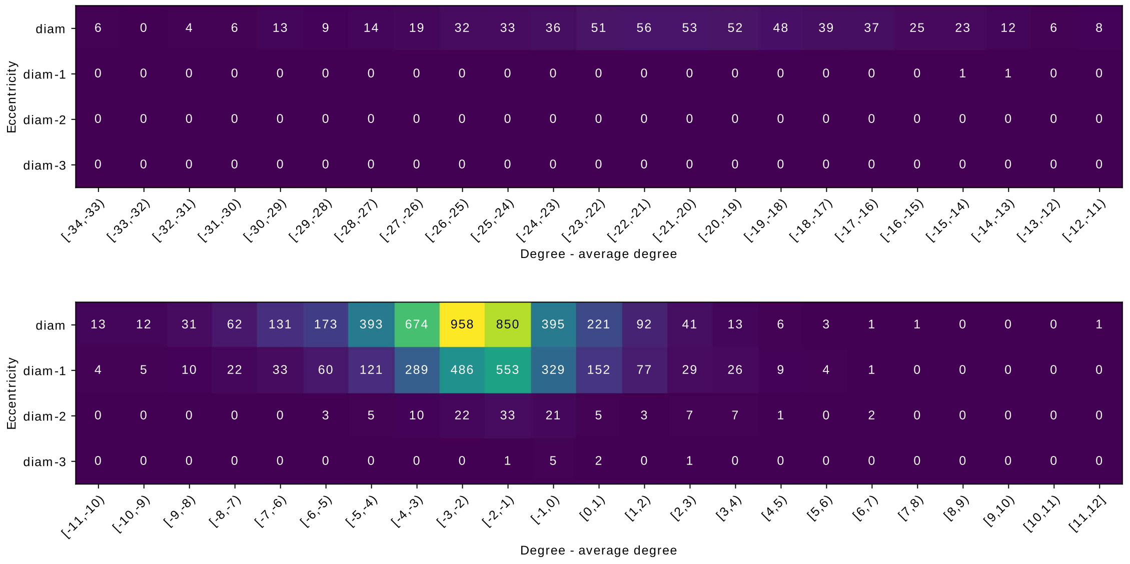

FIGURE 11. Number of extremities of optimal solutions with specific eccentricity and interval of degree difference (over all instances solved by algorithm EXACT-ENUM).

algorithm EXACT-ENUM), as well as the information corresponding to each set of instances. We observe that 66.5% of the distinct extremities of the optimal paths have their eccentricities equal to the graph diameter, while 98.2% have their eccentricities greater than or equal to the diameter less one.

Figure 8 relates the number of extremities of the optimal solutions with their degrees. We provide the global information (considering all test instances exactly solved by algorithm EXACT-ENUM), as well as the information corresponding to each set of instances. We observe that 78.3% of the distinct extremities of the optimal paths have their degrees between 1 and 5, with 90.2% with degree less than or equal to 10. Extremities with degree greater than 15 (8.4% of the total) appear only for instances in the BAL test set, which are those with the largest average degree.

We also evaluated the difference D_{deg} between the degree of the extremities and the average degree. Figure 9 presents the frequency histogram of the degree difference (calculated as the extremity degree minus the average

Table 6. Comparative time-to-target results between the greedy version G-HLIPP and heuristic HLIPP on large graphs.

Name		LH100				LH1000				LH10000		
		HLIPP	G-HLIPP			HLIPP	G-HLIPP			HLIPP	G-HLIPP	
	Target	Time (s)	Time (s)	Ratio	Target	Time (s)	Time (s)	Ratio	Target	Time (s)	Time (s)	Ratio
abyss	6	0.0003	0.0002	0.715	6	0.0003	0.0002	0.878	6	0.0003	0.0002	0.763
hi-tech	13	0.0062	0.0019	0.309	13	0.0355	0.0079	0.221	13	0.0345	0.0071	0.206
karate	9	0.0118	0.0019	0.162	9	0.0132	0.0026	0.197	9	0.0131	0.0026	0.201
mexican	16	0.0059	0.0029	0.488	16	0.0585	0.0107	0.183	16	0.1501	0.0109	0.072
sawmill	18	0.0200	0.0087	0.437	18	0.0026	0.0026	0.995	18	0.0026	0.0022	0.865
pulp fiction	7	0.0130	0.0014	0.109	7	0.0119	0.0014	0.118	7	0.0116	0.0013	0.114
chesapeake	16	0.0231	0.0208	0.901	16	0.2175	0.1389	0.639	16	0.1154	0.0694	0.602
tailorS1	13	0.0011	0.0040	3.719	13	0.0011	0.0026	2.315	13	0.0010	0.0024	2.394
tailorS2	12	0.0030	0.0050	1.657	13	0.0179	0.0384	2.152	15	0.1668	0.1431	0.858
romeo and juliet	9	0.0127	0.0083	0.653	9	0.0099	0.0027	0.275	9	0.0100	0.0032	0.319
oceans 12	8	0.0038	0.0053	1.400	8	0.0089	0.0069	0.771	8	0.0092	0.0070	0.761
die hard	10	0.0046	0.0117	2.549	10	0.0061	0.0114	1.875	10	0.0064	0.0123	1.929
star wars II	8	0.0086	0.0001	0.009	8	0.0143	0.0001	0.005	8	0.0152	0.0001	0.005
oceans 11	8	0.0061	0.0002	0.033	9	0.0338	0.0130	0.386	9	0.0349	0.0127	0.364
the departed	8	0.0059	0.0014	0.245	8	0.0072	0.0016	0.220	8	0.0076	0.0016	0.206
krebs	15	0.0171	0.0422	2.462	17	0.2056	0.3427	1.667	17	0.2374	0.4654	1.960
philadelphia	9	0.0029	0.0073	2.478	10	0.0205	0.0837	4.079	10	0.0210	0.0812	3.871
2012	11	0.0674	0.0550	0.817	12	0.0991	0.0161	0.163	12	0.1091	0.0165	0.151
braveheart	11	0.0003	0.0077	28.657	11	0.0003	0.0365	135.056	11	0.0003	0.0362	121.688
huck	9	0.0766	0.0175	0.229	9	0.0106	0.0119	1.122	9	0.0099	0.0122	1.232
gandhi	9	0.0605	0.0026	0.042	10	0.0356	0.0468	1.314	10	0.0371	0.0439	1.182
watchmen	9	0.0218	0.0002	0.008	9	0.0483	0.0002	0.004	9	0.0465	0.0002	0.004
jean	11	0.0734	0.0047	0.063	11	0.2862	0.0064	0.022	11	0.3983	0.0062	0.016
godfather II	16	0.1159	0.0014	0.012	18	0.0194	–	–	18	0.0188	0.5736	30.586
catch me if you can	8	0.0014	0.0139	10.186	8	0.0013	0.0225	16.874	8	0.0011	0.0225	19.719
david	18	0.0037	0.1002	27.039	19	0.8180	0.2282	0.279	19	1.3102	0.5103	0.390
doors	12	0.2704	0.0078	0.029	12	0.0367	0.0363	0.989	12	0.0361	0.0356	0.984
public enemies	17	0.0254	0.0150	0.589	20	0.1028	0.1097	1.067	20	0.1581	0.2191	1.386
santafe	13	0.0268	0.0003	0.010	13	0.0318	0.0003	0.009	13	0.0321	0.0003	0.009
anna	16	0.3898	0.0916	0.235	20	2.1087	0.1930	0.092	20	5.0350	0.4931	0.098
attiro	28	0.0827	0.0699	0.846	29	0.2321	0.0166	0.072	30	1.8867	2.9333	1.555
dolphins	20	0.0106	0.0013	0.124	22	0.8837	0.2206	0.250	23	0.0866	0.7646	8.827
prison	28	0.0795	0.0161	0.202	31	0.7005	0.0834	0.119	36	1.9405	0.0460	0.024
sanjuansur	35	0.0832	0.0179	0.216	35	0.6871	0.1091	0.159	37	11.7761	4.0833	0.347
ieeebus	44	0.2010	0.2563	1.275	44	1.6414	2.2618	1.378	47	4.4927	1.8234	0.406
USAir97	29	0.1148	0.0393	0.343	33	4.3122	2.4855	0.576	38	110.4310	169.3410	1.533
494_bus	101	1.3763	–	–	107	12.2402	6.2169	0.508	114	332.5640	7.3518	0.022
662_bus	242	15.3271	21.9074	1.429	242	94.1874	141.2080	1.499	242	887.9110	1332.9700	1.501
S.Cerevisae	144	0.6410	30.0306	46.849	149	152.3800	30.9581	0.203	155	1806.8500	2133.2800	1.181
Number of times G-HLIPP was faster than HLIPP:				26/39				26/39				24/39

degree), for each set of test instances and for all solved instances. Comparing Figures 8 and 9 for the complete set of instances, we notice that in the second there is a reduction in the range of values represented in the horizontal axis with a frequency greater than zero. This might be useful in a heuristic to cut the subset of vertices that will be explored as extreme vertices for optimal solutions. We observe that 73.6% of the distinct extremities of optimal solutions have $D_{\text{deg}} \in [-5, 0)$, with 88.4% of them with $D_{\text{deg}} \in [-8f, 2)$. Furthermore, 89.9% of the extremities have their degrees smaller than the average degree.

Considering all test instances solved by algorithm EXACT-ENUM, Figure 10 shows a heatmap representing the number of extremities of optimal solutions with specific eccentricity and degree. We notice that 51.8% of the distinct extremities of optimal solutions have their eccentricity equal to the graph diameter and degree less than or equal to five, with 88.6% with their eccentricity greater than or equal to the diameter less one and degree less than or equal to ten.

Similarly, for the same test instances solved by algorithm EXACT-ENUM, Figure 11 shows a heatmap representing the number of extremities of optimal solutions with specific eccentricity and interval of degree difference. The degree difference was calculated as the vertex degree less the average degree of the graph. In this case, we observe that 46.8% of the distinct extremities of optimal solutions have their eccentricity equal to the graph diameter and $D_{\text{deg}} \in [-5, 0)$, with 86.8% with their eccentricity greater than or equal to the diameter less one and $D_{\text{deg}} \in [-8, 2)$.

These two heatmaps clearly indicate that both the eccentricity and the degree play an important role to characterize the extremities of optimal solutions to the longest induced path problem.

These results and investigations are useful to guide the search for optimal solutions and better algorithms. An improved greedy version G-HLIPP of Algorithm 3 was developed in which vertices are selected in the non-increasing order of their eccentricities. Ties are broken in favor of the vertex with smaller degree.

Table 6 compares the two heuristics HLIPP and G-HLIPP in terms of the time they take to find a target solution value. The targets are the best values obtained by heuristic HLIPP in Table 3. For each stopping criterion (*i.e.*, LH100, LH1000, and LH10000), we indicate the ratio between the times taken by G-HLIPP and HLIPP. Ratios smaller (resp. greater) than one correspond to instances where the greedy heuristic G-HLIPP was faster (resp. slower) than HLIPP. We observe that this simple improvement reduced the times to target values for 65% of the runs (76 out of 117). The average time-to-target reduction between G-HLIPP and HLIPP for these runs was 0.315. On the other hand, for the remaining runs where HLIPP was still faster, the average time-to-target reduction between HLIPP and G-HLIPP was 0.459. We observe that for two runs G-HLIPP did not find the target before the stopping criterion.

6. Concluding remarks

In this article, we first proposed an exact enumerative algorithm based on backtracking for the problem of finding the longest induced path in a graph. The new algorithm is faster than that of Matsypura *et al.* [13] and competitive with the best ILP_{Cut} implementations of Bökler *et al.* [5].

We also developed a new heuristic for the problem, which explores all vertices of the graph as possible source vertices of induced paths. Computational results showed that the newly proposed heuristic was consistently faster and more robust than the randomized heuristic of Matsypura *et al.* [13] on the same test problems.

The computational experiments also presented a numerical study correlating the eccentricity and the degree of the vertices with the frequency in which they appear as extremities of optimal solutions, *i.e.*, longest induced paths. These results lead to the implementation of an improved version of the heuristic, in which the vertices are explored in the non-increasing order of their eccentricities.

Acknowledgements. The authors are grateful to Mark Newman for sending the dataset of the Santa Fe Institute collaboration network. Work of Ruslán G. Marzo was supported by a CAPES scholarship and by FAPERJ scholarship E-26/200.330/2020. Work of Celso C. Ribeiro was partially supported by CNPq research grants 303958/2015-4 and 425778/2016-9 and by FAPERJ research grant E-26/202.854/2017. This work was also partially sponsored by Coordenação de Aperfeioamento de Pessoal de Nível Superior (CAPES), under Finance Code 001.

References

[1] R.K. Ahuja, T.L. Magnanti and J.B. Orlin, Network flows. Elsevier (1988).

[2] P. Alles and S. Poljak, Long induced paths and cycles in Kneser graphs. *Graphs Comb.* **5** (1989) 303–306.

[3] A.-L. Barabási and R. Albert, Emergence of scaling in random networks. *Science* **286** (1999) 509–512.

[4] B. Bergougnoux and M.M. Kanté, A meta-algorithm for solving connectivity and acyclicity constraints on locally checkable properties parameterized by graph width measures. Preprint `arXiv:1805.11275` (2018).

[5] F. Bökler, M. Chimani, M.H. Wagner and T. Wiedera, An experimental study of ILP formulations for the longest induced path problem. In Combinatorial Optimization, edited by M. Baïou, B. Gendron, O. Günlük and A.R. Mahjoub. In Vol. 12176 of *Lecture Notes in Computer Science*. Springer, Cham (2020).

[6] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman and Co., New York (1979).

[7] F. Gavril, Algorithms for maximum weight induced paths. *Inf. Process. Lett.* **81** (2002) 203–208.

[8] T. Ishizeki, Y. Otachi and K. Yamazaki, An improved algorithm for the longest induced path problem on k-chordal graphs. *Discrete Appl. Math.* **156** (2008) 3057–3059.

[9] L. Jaffke, O.-J. Kwon and J. Telle, Polynomial-time algorithms for the Longest Induced Path and Induced Disjoint Paths problems on graphs of bounded mim-width. In: Vol. 89 of *Leibniz International Proceedings in Informatics* (2018) 1–21.

[10] L. Jaffke, O.-J. Kwon and J.A. Telle, Mim-Width I. Induced path problems. *Discrete Appl. Math.* **278** (2020) 153–168.

[11] W.H. Kautz, Unit-Distance Error-Checking Codes. *IRE Trans. Electron. Comput.* **7** (1958) 179–180.

[12] D. Kratsch, H. Müller and I. Todinca, Feedback vertex set and longest induced path on AT-free graphs. In: International Workshop on Graph-Theoretic Concepts in Computer Science. Springer (2003) 309–321.

[13] D. Matsypura, A. Veremyev, O.A. Prokopyev and E.L. Pasiliao, On exact solution approaches for the longest induced path problem. *Eur. J. Oper. Res.* **278** (2019) 546–562.

[14] Moviegalaxies, Social networks in movies. Available from: https://moviegalaxies.com/ (2020).

[15] NetworkX developers, NetworkX documentation. Available from: http://networkx.github.io/ (2020).

[16] M.E. Newman, The structure and function of complex networks. *SIAM Rev.* **45** (2003) 167–256.

[17] PassMark, PassMark Software – CPU Benchmarks. Intel Core i5-4460S *vs.* Intel Xeon E5-1650 v2 *vs.* Intel Xeon Gold 6134. Available from: https://www.cpubenchmark.net/compare/Intel-i5-4460S-vs-Intel-Xeon-E5-1650-v2-vs-Intel-Xeon-Gold-6134/2232vs2066vs3008 (2020).

[18] K. Pearson, Notes on regression and inheritance in the case of two parents. *Proc. R. Soc. London* **58** (1895) 240–242.

[19] L.O. Prokhorenkova and E. Samosvat, Global clustering coefficient in scale-free networks. In: *International Workshop on Algorithms and Models for the Web-Graph.* Springer (2014) 47–58.

[20] D.J. Watts and S.H. Strogatz, Collective dynamics of “small-world” networks. *Nature* **393** (1998) 440–442.