Computational Methods for Finding Long Simple Cycles in Complex Networks

(This is an author copy of the accepted version of the article published in Knowledge-based Systems, 125, 96–107, `http://dx.doi.org/10.1016/j.knosys.2017.03.022`)

David Chalupa, Phininder Balaghan, Ken A. Hawick, Neil A. Gordon

Computer Science
School of Engineering and Computer Science
University of Hull
Cottingham Road
Hull HU6 7RX, United Kingdom
Email: { `d.chalupa, p.balaghan, k.a.hawick, n.a.gordon` }`@hull.ac.uk`

Abstract

Detection of long simple cycles in real-world complex networks finds many applications in layout algorithms, information flow modelling, as well as in bioinformatics. In this paper, we propose two computational methods for finding long cycles in real-world networks. The first method is an exact approach based on our own integer linear programming formulation of the problem and a data mining pipeline. This pipeline ensures that the problem is solved as a sequence of integer linear programs. The second method is a multi-start local search heuristic, which combines an initial construction of a long cycle using depth-first search with four different perturbation operators. Our experimental results are presented for social network samples, graphs studied in the network science field, graphs from DIMACS series, and protein-protein interaction networks. These results show that our formulation leads to a significantly more efficient exact approach to solve the problem than a previous formulation. For 14 out of 22 networks, we have found the optimal solutions. The potential of heuristics in this problem is also demonstrated, especially in the context of large-scale problem instances.

Keywords:

long simple cycles, long cycles, complex networks, integer linear programming, graph algorithms, local search, Hamiltonian cycles

1. Introduction

With the growing volume of information available as connected data, the need for *efficient algorithms to solve graph problems* has been becoming increasingly important [20].

The longest simple cycle problem is one of the classical NP-hard graph problems, in which one simply aims to find the longest simple cycle in an undirected graph. A simple cycle is defined as a connected subgraph of the graph with all vertices having degree 2 and without repetitions in vertices or edges. For simplicity, the term longest cycle will refer to the longest simple cycle hereafter.

A cycle which spans all vertices of the graph is called a Hamiltonian cycle. Hence, the longest cycle problem represents a generalisation of the decision problem for Hamiltonian cycle [26] which is widely studied [6, 15, 44].

Searching for the longest cycle in a graph has been a subject of study in statistical mechanics, with approaches based on message passing and Monte Carlo procedures being used [33]. However, studying the longest cycle problem in the context of both exact and experimental algorithms, as well as applications to real-world complex networks, has still been somewhat limited.

On the other hand, identifying long cycles finds its applications in *automatic drawing of planar graphs* [40] or in *layout algorithms for metabolic pathways* in bioinformatics [5]. It is also closely related to community structure [31], its hierarchy [11] and propagation processes [50] in real-world networks. To the best of our knowledge, the longest cycle problem has not yet been studied in the context of social networks nor several types of biological networks. It seems that a spectrum of potential applications has not been explored yet.

Previous scientific results on the longest cycle problem have been overwhelmingly concerned with its theoretical properties. Complexity and approximability of the problem have been studied in sparse graphs [16, 37], bounded degree graphs [12], triangle-free graphs [3], and small graph classes [43].

Superpolylogarithmically long cycle detection [17], treewidth-based approximation [2, 9], and matrix-based approximation [34] have been explored

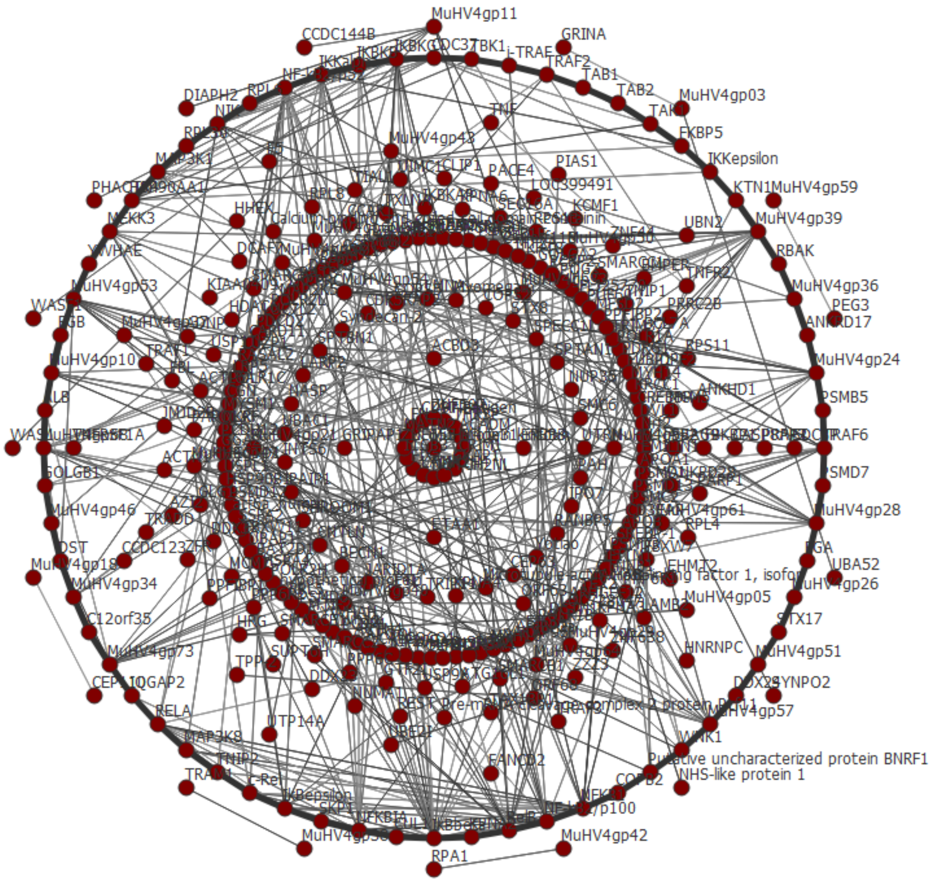

Figure 1: The longest cycle identified for a high-throughput human protein-protein interaction network *Hsapi20160114HT* from the UCLA database of interacting proteins [38, 47, 48, 49]. Both of our approaches have found this cycle which consists of 64 vertices.

as avenues for efficient finding of approximate solutions to the problem.

Exact methods for the problem include enumerative techniques [21, 23, 41] and a branch-and-bound algorithm, which is based on an integer linear programming (ILP) formulation of the problem [14]. The enumerative techniques allow one to explore the distribution of the cycle numbers, as a function of the cycle length. This idea has successfully been explored for small worlds, as well as the Kauffman networks [22, 28]. However, these algorithms are naturally computationally demanding. Therefore, for large graphs, it is interesting to search for the location of the far end of this distribution using the identification of the longest cycle.

For the Hamiltonian cycle problem, approximation algorithms for maximal planar graphs have been explored [36]. Heuristics for the problem are a subject of interest, too. These include an ant-inspired heuristic [44] or an interior point heuristic [15]. Hamiltonian cycles in scale-free networks have also been previously explored [6]. The longest cycle problem is also often linked with the longest path problem, for which efficient algorithms [42] and approximations [25] have been studied. Some other studies have also been concerned with directed longest cycles [8].

Figure 1 shows the longest cycle in a high-throughput human protein-protein interaction network, taken as one of the test instances for this paper from the UCLA database of interacting proteins [38, 47, 48, 49]. This cycle consists of 64 vertices. This visualisation arranges the proteins of the longest cycle as an outer ring of the drawing. This shows the long cyclic dependency of the protein interactions. These are valuable in a reconstruction of the metabolic pathways [13], in which long cycles are used in layout algorithms for their visualisation [5].

In this present article we focus on computational methods for finding long cycles in real-world complex networks. These include social networks, protein-protein interactions, as well as networks from several other domains. Our approaches include both an *exact method* based on our own ILP formulation of the problem, as well as a heuristic combining *depth-first search* with a *local improvement procedure* using clustering properties of real-world networks.

Our main contribution is in proposing a new ILP formulation the longest cycle problem, as well as designing of a pipeline for efficient exact approach to find the longest cycle. Additionally, we propose a hybrid heuristic, which combines construction of a long cycle using depth-first search with four local search operators to improve this initial cycle. Our computational results are presented for a diverse set of real-world networks, indicating that both the size and the structure of the graph seem to influence the difficulty of the problem.

Our article is structured as follows. In Section 2 we review the existing ILP formulation of the longest cycle problem with fixed initial vertex and introduce our own ILP formulation of the same problem. We then prove the correctness of our formulation and propose a pipeline for the exact approach which can be used with both formulations. In Section 3 we describe our heuristic based on depth-first search and four local search operators. In Section 4 we present the computational results of the exact approach and compare it to the approach based on the previous formulation. We also show the result of the heuristic on a real-world network data set. In Section 5 we discuss the obtained results and their interpretation. We offer several conclusions and identify the open problems in Section 6.

2. Exact Approach to the Longest Simple Cycle Problem

In this section, we first review the current ILP formulation of the longest simple cycle problem with fixed initial vertex [14]. Next, we introduce our own formulation of this problem. Last but not least, we design a pipeline for efficient solving of the problem as a sequence of ILP problems.

2.1. Dixon and Goodman's Formulation of the Longest Simple Cycle Problem

In the following, we review the formulation used to design the branch-and-bound algorithm for the problem [14]. In this formulation, the longest cycle containing a fixed vertex is considered. The formulation uses a simple trick of introducing a dummy vertex, which has the same adjacencies as the fixed vertex. Instead of searching for the longest cycle, one can then search for the longest path between the fixed vertex and this dummy vertex.

Dixon and Goodman's ILP formulation of the longest cycle problem with a fixed initial vertex [14]. Define an undirected graph $G = [V, E]$ without loops, with vertices $v_1, v_2, ..., v_n$, and assume that the starting vertex for our cycle is known. The known starting vertex is referred to as v_1. Consider a graph obtained from G by adding vertex v_{n+1}, which is a "copy" of vertex v_1, i.e. has the same adjacencies. Let $x_{ij} \in \{0, 1\}$ represent the transitions from the vertex v_i to v_j, i.e. $x_{ij} = 1$ whenever the edge $\{v_i, v_j\}$ is in the cycle and $x_{ij} = 0$ otherwise. Then, we solve the following problem to obtain the longest cycle, which includes v_1:

$$\max \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}, \tag{1}$$

where $c_{ij} = 1$ if vertices v_i and v_j are adjacent and $c_{ij} = 0$ otherwise:

$$c_{ij} = \begin{cases} 1 & \{v_i, v_j\} \in E \\ 0 & \{v_i, v_j\} \notin E \end{cases}, \tag{2}$$

subject to:

$$\sum_{i=1}^{n+1} c_{ik} x_{ik} = F_k, \quad k = 2, 3, ..., n, \tag{3}$$

$$\sum_{i=1}^{n+1} c_{kj} x_{kj} = F_k, \quad k = 2, 3, ..., n, \tag{4}$$

$$\sum_{j=2}^{n} c_{1j} x_{1j} = \sum_{i=1}^{n} c_{i(n+1)} x_{i(n+1)} = 1, \tag{5}$$

$$y_i - y_j + n x_{ij} \leq n - 1, \quad i, j = 1, 2, ..., n+1, \tag{6}$$

$$y_i \leq n + 1, \quad i = 1, 2, ..., n+1, \tag{7}$$

where:

$$F_i = \begin{cases} 1 & if\ v_i\ is\ used\ in\ the\ longest\ cycle \\ 0 & otherwise \end{cases}. \tag{8}$$

In this formulation, y_i are dummy integer variables introduced for each vertex v_i. Constraints (6) and (7) ensure that one cycle is detected, instead of a set of disjoint cycles. The authors of this formulation report that it leads to a highly constrained feasible region of the search space and to a low efficiency already for graphs with tens of vertices. More particularly, their experimental results were presented for graphs on at most 40 vertices. In our computational results, we will demonstrate that it is possible to formulate the problem more efficiently.

2.2. Our Formulation of the Longest Simple Cycle Problem

In the following, we present our own ILP formulation of the longest cycle problem. Unlike the previous formulation based on a few tricks, our formulation is based on an adaptation of a flow-based formulation of the travelling salesperson problem [18].

Our ILP formulation of the longest cycle problem with a fixed initial vertex. Define an undirected graph $G = [V, E]$ without loops, with vertices $v_1, v_2, ..., v_n$, and assume that the starting vertex for our cycle is known. The known starting vertex is referred to as v_1. Let $x_{ij} \in \{0, 1\}$ represent the transitions from the vertex v_i to v_j, i.e. $x_{ij} = 1$ whenever the edge $\{v_i, v_j\}$ is in the cycle and $x_{ij} = 0$ otherwise. Then, we solve the following problem to obtain the longest cycle, which includes v_1:

$$\max \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}, \tag{9}$$

where $c_{ij} = 1$ if vertices v_i and v_j are adjacent and $c_{ij} = 0$ otherwise:

$$c_{ij} = \begin{cases} 1 & \{v_i, v_j\} \in E \\ 0 & \{v_i, v_j\} \notin E \end{cases}, \tag{10}$$

subject to:

$$\sum_{i=1}^{n} x_{ij} - \sum_{i=1}^{n} x_{ji} = 0, \quad j = 1, 2, ..., n,\ \{v_i, v_j\} \in E, \tag{11}$$

$$\sum_{i=1}^{n} x_{ij} + \sum_{i=1}^{n} x_{ji} \leq 2, \quad j = 1, 2, ..., n,\ \{v_i, v_j\} \in E, \tag{12}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \geq 3,\ \{v_i, v_j\} \in E, \tag{13}$$

$$y_{ij} \leq (n-1) x_{ij}, \quad i = 2, 3, ..., n,\ j = 1, 2, ..., n,\ \{v_i, v_j\} \in E, \tag{14}$$

$$2 \sum_{j=1, j \neq i} y_{ij} - 2 \sum_{j=2, j \neq i} y_{ji} - \sum_{j=1, j \neq i} x_{ij} - \sum_{j=2, j \neq i} x_{ji} = 0, \tag{15}$$

$$i = 2, 3, ..., n,\ \{v_i, v_j\} \in E,$$

where $y_{ij} \geq 0$. Values y_{ij} represent a flow on the edges of the graph, which is used to ensure that the resulting subgraph is a single cycle, rather than a union of several disjoint cycles. It is straightforward to adapt this ILP program to cases with the starting vertex being different to v_1. In the next step, we prove that this ILP formulation corresponds to the problem of the longest cycle containing v_1.

Theorem. Let the vertices of an undirected graph $G = [V, E]$ without loops be $v_1, v_2, ..., v_n$. Then, our ILP formulation of the longest simple cycle problem with a fixed initial vertex corresponds to the problem of finding the longest simple cycle $C = [V_C, E_C]$ such that $v_1 \in V_C$.

Proof. Let $C = [V_C, E_C]$ be a simple cycle of length k in graph G, with vertices ordered $v_{c_0} = v_1$, v_{c_1}, v_{c_2}, ..., $v_{c_{k-1}}$. Then, the first three constraints of the ILP program are met, since for each $v \in V_C$, there is exactly one transition from v and one transition to v in our ordering. There are no transitions from and to v_i if $v_i \notin V_C$, which meets the constraints (11) and (12), too. We have that $k \geq 3$, i.e. the constraint (13) is met.

Consider a flow on the edges of G such that $y_{c_i c_{i+1}} = i$ for each $i = 0, 1, 2, ..., k-2$ and $y_{ij} = 0$ if $\{v_i, v_j\} \notin E_C$. Then, the capacity constraint (14) is true, since $k - 1 \leq n - 1$. For the constraint (15), consider the transitions for a vertex $v_i \in V_C$ first. Let $v_i = v_{c_\ell}$, i.e. let it be the $(\ell + 1)$-st vertex in the ordering of vertices in the cycle. Then, the difference in the flow out and in the vertex is:

$$\sum_{j=1, j\neq i} y_{ij} - \sum_{j=2, j\neq i} y_{ji} = \ell - (\ell - 1) = 1, \tag{16}$$

which is equal to:

$$\frac{1}{2}\left[\sum_{j=1, j\neq i} x_{ij} + \sum_{j=2, j\neq i} x_{ji}\right] = \frac{1}{2}(1 + 1) = 1. \tag{17}$$

This proves that every simple cycle in G meets the constraints of our ILP program.

For the inverse implication, we have that the constraints (11) and (12) can only be met if for each $i = 1, 2, ..., n$, one of two situations takes place.

The first case holds if there is exactly one j' and one j'' such that $x_{ij'} = 1$ and $x_{j''i} = 1$, with other values x_{ij} and x_{ji} being zero for our vertex v_i and for $j \notin \{j', j''\}$.

The second case is that all values x_{ij} and x_{ji} are zero for vertex v_i. If we assume that $x_{ij'} = 1$ and $x_{j''i} = 0$ for some j' and j'', this could satisfy constraint (12) but would not satisfy constraint (11).

Constraint (13) only ensures that there are at least three chosen transitions which holds for all simple cycles. This already implies that an assignment of values to x_{ij} meeting the first three constraints leads to a simple cycle in G or a union of multiple disjoint simple cycles in G.

For the flow-based constraints, let us consider a union of two disjoint simple cycles of lengths k and t such that $k + t \leq n$ and vertex v_1 belongs to the cycle of length k. Consider the ordering $v_{c_0} = v_1, v_{c_1}, v_{c_2}, ..., v_{c_{k-1}}$ of vertices in the cycle of length k. Then, a flow $i \leq y_{c_i c_{i+1}} \leq n - 1$, with each vertex incrementing the flow by one for each $i = 0, 1, ..., k - 2$, will fulfil both of the flow-based constraints, similarly to the arguments above.

At this point, consider the simple cycle of length t, with the ordering $v_{c_k}, v_{c_{k+1}}, ..., v_{c_{k+t-1}}$ of its vertices. Then, we have that $i \leq y_{c_{k+i} c_{k+i+1}} \leq n - 1$ represent the feasible values of flow for this cycle, with each vertex incrementing the flow by one. However, this means that the difference between the incoming and the outcoming flow for v_{c_k} will be:

$$\sum_{j=1, j \neq i} y_{c_k j} - \sum_{j=2, j \neq i} y_{j c_k} = t - 1 > 1, \tag{18}$$

since each vertex on the cycle contributes to the flow by one. This is in contradiction with our premises, since only v_1 is exempt from this constraint. ■

This already establishes that our ILP formulation is correct for the restricted variant of the longest simple cycle problem. The general longest simple cycle problem will then be solved as a sequence of consecutive ILP problems with different fixed initial vertices. As the problem is NP-hard, computational complexity of this approach is exponential in the worst case. There are 2^m possible feasible and infeasible assignments of binary values to x_{ij}, where m is the number of edges in the graph.

It is worth mentioning that it is also possible to extend the formulation presented above to obtain a general ILP for the longest simple cycle problem. However, our preliminary experiments indicated that the ILP solver seems to be more efficient if the sequence of elementary ILP problems are solved, rather than one large ILP problem. This is probably due to the fact that large infeasible regions are induced in the search space by the addition of more constraints.

The general formulation uses the idea of introducing a dummy vertex as a source for the flow and the cycle and searches for the longest cycle from the source. It follows from the assumption that the longest cycle of the graph with the source vertex can be transformed into the longest cycle of the original graph if and only if the neighbours of the source vertex on the cycle are adjacent. Therefore, we are searching for the longest simple cycle of the graph with the source vertex such that the source vertex, and its neighbours on the cycle form a triangle.

Let the source vertex be v_0 and let it be adjacent to all other vertices. Then, one can solve the ILP problem above on the graph with the initial vertex v_0 and introduce dummy variables $z_{ij} \in \{0,1\}$. These variables represent the choice of the single edge, which ensures that our longest cycle in the modified graph can be transformed into the longest cycle of the original graph by substituting transitions x_{0j} and x_{i0} with transition x_{ij}, for which $z_{ij} = 1$. The additional constraints, which assure that v_0, v_i and v_j form a triangle, are the following:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} z_{ij} = 1, \tag{19}$$

$$x_{0j} + x_{i0} - 2z_{ij} \geq 0, \quad i,j = 1,2,3,...,n. \tag{20}$$

Note that the optimal value of the objective function for this formulation is actually higher than the length of the longest cycle by 1, due to the introduction of the dummy vertex.

2.3. Pipeline Design for the Exact Approach

A general formulation of the longest cycle problem was created. We have found that this approach was less efficient at finding the longest cycle in comparison to creating a sequence of ILP programs for fixed initial vertices. Therefore a pipeline was created to handle this sequence.

Figure 2 shows the process describing the data mining pipeline. We begin by first pruning the leaves of the graph, as these cannot form a cycle. This pruning is iterated until the graph contains no leaves. We then check for any

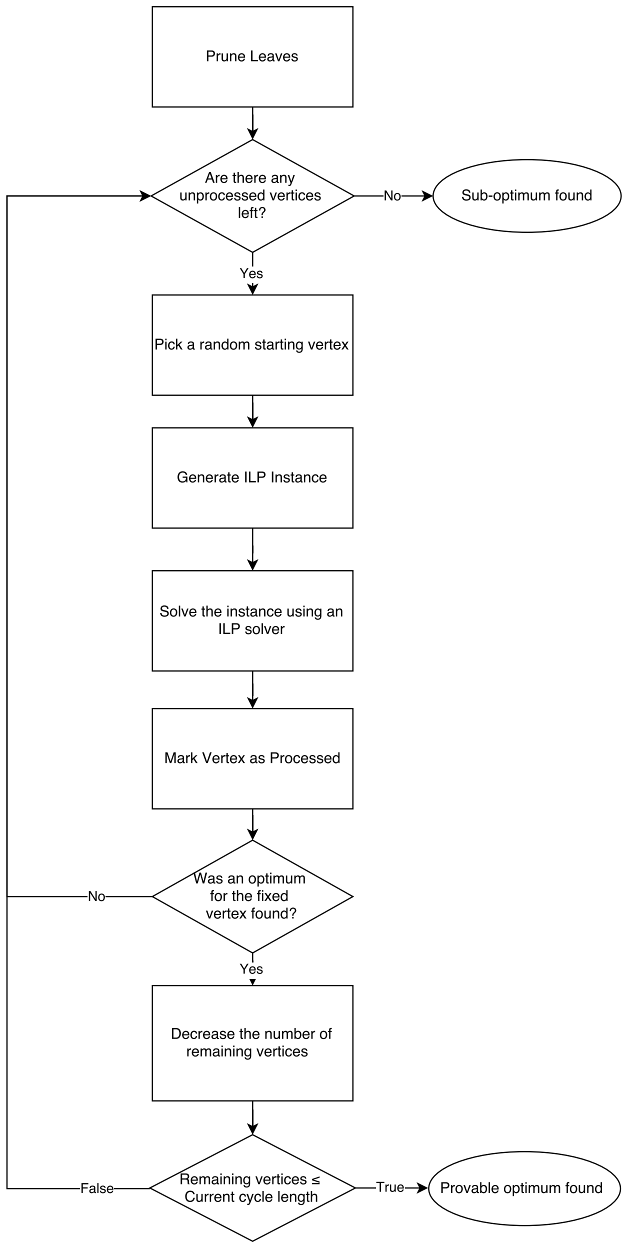
Prune Leaves
Are there any unprocessed vertices left?
No
Sub-optimum found
Yes
Pick a random starting vertex
Generate ILP Instance
Solve the instance using an ILP solver
Mark Vertex as Processed
Was an optimum for the fixed vertex found?
No
Yes
Decrease the number of remaining vertices
Remaining vertices ≤ Current cycle length
False
True
Provable optimum found

Figure 2: Our process of the search for the longest cycle using an exact approach based on fixed initial vertex. The process starts by pruning the leaves of the graph and follows by generating a sequence of ILP instances for randomly chosen fixed initial vertices.

unprocessed vertices, should none exist then a suboptimal cycle has been found. If an unprocessed vertex has been found, we pick a random vertex from the remaining unprocessed vertices to be our fixed initial vertex.

We then generate an ILP instance with the fixed initial vertex based on our formulation or Dixon and Goodman's formulation. The instance is solved by an ILP branch-and-cut solver. Optimality of the output is checked. The initial vertex is then marked as processed. If an optimal solution has not been found, we begin to process the next available vertex, should one exist. However if an optimal solution has been found, we check if its length is higher than the highest length found so far, and possibly replace the current best cycle. We then decrease the amount of remaining vertices to process. This amount is then compared to the current longest cycle length. If it is less than the current longest cycle length, a provable optimum has been found and the process terminates. Otherwise we continue to process the next available vertex, should one exist.

3. Heuristic Methods for the Longest Cycle Problem

Apart from the exact approaches, approximation algorithms and heuristics are of a high interest for detection of long cycles in the more difficult problem instances. Some approximation algorithms use recursive disjoint cycle detection procedures [7]. Other algorithms employ detection of cycles to enlarge other cycles. This makes the methods possible to be used in an iterated way [17].

Some earlier approximation algorithms have used two phases, combining depth-first search (DFS) for sampling of the initial solution with dynamic programming as the second phase [9].

In this section we present an algorithm which uses DFS to sample an initial long cycle, with four perturbation operators used in a local search to enlarge this cycle. DFS is used to sample long paths and filter those which can be "closed" to cycles. In each branching step of DFS the neighbours are scanned in a random order. This allows the usage of DFS as a randomised algorithm.

The pseudocode of our version of DFS is given in Algorithm 1. In step 1, we start with an empty cycle C and set the initial cycle length l to 0. An iterative procedure over all vertices follows. Let $v_{central}$ be the vertex, from which we start constructing the cycle. We use a stack of vertices S and functions $d(v)$ and $p(v)$ which denote the distance of v from $v_{central}$ and a binary value, which indicates if v has already been processed by DFS. In step 6, we pop the current vertex from S and process it if it has not been processed within DFS yet. The neighbours of v are scanned in step 9 in a random order. Let w be the current neighbour in this scanning. Then, steps 10-13 are used to ensure that if the path from $v_{central}$ through v to w is longer than the one previously sampled, then w is moved to the top of the stack S. This makes the algorithm prefer longer paths. Also, we use array *parent* to record predecessors of vertices in DFS, to be able to trace the final long cycle. In steps 14-16, we update the current longest cycle C if $v_{central}$ was reached once again and its length d_c is the highest length found so far.

When the DFS is finished, we use a local search approach to enlarge the initial long cycle. Four perturbation operators are used. Two of these operators serve as improvement operators which enlarge the current cycle by substituting subpaths of length 1 with subpaths of length 2 or 3. Two other operators represent the plateau exploration operators which substitute paths of lengths 2 or 3 by paths of the same length. All four perturbation operators are depicted in Figure 3.

In the following, we will define the the four perturbation operators formally. However, let us first formulate a few common terms for all operators. Let $C = [v_{c_1}, v_{c_2}, ..., v_{c_k}]$ be a cycle of length k on vertices of graph $G = [V, E]$. Let V_C be the set of vertices in C and let E_C be the set of edges in C. Then, the perturbation operators will be defined as follows.

Perturbation operator 1 (triangular improvement operator). If there is a vertex $w \in V \backslash V_C$ such that for some i it holds that $\{v_{c_i}, v_{c_{i+1}}\} \in E_C$ and vertices $v_{c_i}, w, v_{c_{i+1}}$ form a triangle, then G also contains a cycle $C' = \{v_{c_1}, v_{c_2}, ..., v_{c_i}, w, v_{c_{i+1}} ..., v_{c_k}\}$ of length $k + 1$.

Perturbation operator 2 (rectangular improvement operator). If there are vertices $w_1, w_2 \in V \backslash V_C$ such that $\{w_1, w_2\} \in E$ and for some i it holds that $\{v_{c_i}, v_{c_{i+1}}\} \in E_C$ and the sequence $[v_{c_i}, w_1, w_2, v_{c_{i+1}}]$ represents a cycle of length 4, then G also contains a cycle $C' = \{v_{c_1}, v_{c_2}, ..., v_{c_i}, w_1, w_2, v_{c_{i+1}} ..., v_{c_k}\}$ of length $k + 2$.

Perturbation operator 3 (plateau exploration operator 1). If there is a vertex $w \in V \backslash V_C$ such that for some i it holds that $\{v_{c_i - 1}, v_{c_i}\} \in E_C$ and $\{v_{c_i}, v_{c_{i+1}}\} \in E_C$, then if $\{v_{c_i - 1}, w\} \in E$ and $\{w, v_{c_{i+1}}\} \in E$ as well, then G also contains a cycle $C' = \{v_{c_1}, v_{c_2}, ..., v_{c_{i-1}}, w, v_{c_{i+1}}, ..., v_{c_k}\}$ of length k.

Algorithm 1. Identifying long cycles using Depth-first Search (DFS)

```
     Input: graph $G = [V, E]$
     Output: cycle $C = [v_{c_1}, v_{c_2}, ..., v_{c_k}]$
1    $C = []$, $l = 0$
2    for each $v_{central} \in V$
3        $\forall v \in V$ let $d(v) = 0$, $p(v) = 0$
4        $S = [v_{central}]$
5        while $S$ contains at least one vertex
6            $v = pop(S)$
7            if $p(v) = 0$
8                $d_c = d(v)$, $p(v) = 1$
9                for each $w$ such that $\{v, w\} \in E$ in
                 a *random order*
10                   if $p(w) \neq 0 \wedge w \neq v_{central} \wedge d(w) \leq d_c + 1$
11                       remove $w$ from $S$ if it currently is in $S$
12                       $S = push(w)$
13                       $d(w) = d_c + 1$, $parent(w) = v$
14                   if $w = v_{central} \wedge d_c \geq 2 \wedge l < d_c + 1$
15                       $l = d_c + 1$
16                       use array *parent* to trace and update
                         the current longest cycle $C$ to $v_{central}$
17   return $C$
```

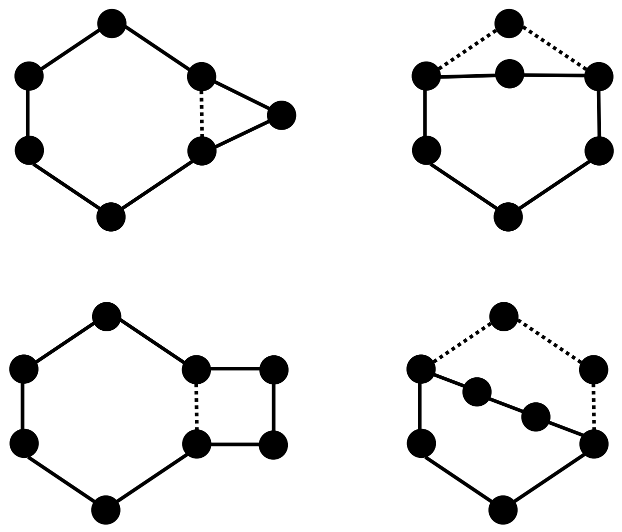

Figure 3: Perturbation operators used in our local search algorithm for the longest cycle problem. On the left-hand side, the perturbation operators 1 (triangular improvement operator, above) and 2 (rectangular improvement operator, below) are depicted. If there is an edge on the cycle, which lies in a triangle or a rectangle with the corresponding vertices lying outside of the cycle, then the edge can be substituted by the two or three edges. On the right-hand side, we illustrate the perturbation operator 3 (the plateau exploration operator 1, above) and the perturbation operator 4 (the plateau exploration operator 2, below). In these cases, a subpath of the cycle on three or four vertices is substituted by an alternative subpath with the same number of vertices.

Perturbation operator 4 (plateau exploration operator 2). If there are vertices $w_1, w_2 \in V \backslash V_C$ such that $\{w_1, w_2\} \in E$ and for some i it holds that $\{v_{c_{i-2}}, v_{c_{i-1}}\} \in E_C$, $\{v_{c_{i-1}}, v_{c_i}\} \in E_C$ and $\{v_{c_i}, v_{c_{i+1}}\} \in E_C$, then if $\{v_{c_{i-2}}, w_1\} \in E$, and $\{w_2, v_{c_{i+1}}\} \in E$ as well, then G also contains a cycle $C' = \{v_{c_1}, v_{c_2}, ..., v_{c_{i-2}}, w_1, w_2, v_{c_{i+1}}, ..., v_{c_k}\}$ of length k.

These perturbation operators effectively use the properties of real-world complex networks. One of the properties being used is often represented by the clustering coefficient metric, which is defined as a ratio of the number of triangles to the number of all connected triplets of vertices [1]. Many real-world networks have relatively high clustering coefficients, which means that many connected triplets form triangles in these networks. This holds both for the lattice-based small worlds such as the networks generated by the Watts-Strogatz model [45, 46], as well as the scale-free networks generated by models such as the Barabási-Albert preferential attachment model [4]. This observation supports the use of the triangular perturbation operator. On the other hand for grid-based networks, the rectangular operator seems to be a much more valid choice. The underlying structure tends to be regular mesh, for which the triangular operator will not be suitable.

The plateau exploration operators also use the properties of complex networks. Perturbation operator 3 substitutes one connected triplet in the cycle by an alternative connected triplet. Perturbation operator 4 represents a similar idea of "second order", substituting a connected 4-tuple by an alternative connected 4-tuple.

4. Selected Results

In this section, we present the results selected from the extensive computational experiments we performed. In the following, we will first focus on the general experimental design. Next, we present the numerical results of our exact approach, its comparison to the approach based on Dixon and Goodman's formulation [14], as well as numerical results obtained by the heuristic.

4.1. Experimental Design

For the exact approaches, we have used the open-source branch-and-cut ILP solver CBC from the COIN-OR package [10, 30]. The problem was solved as a sequence of ILP programs generated for different starting vertices. Each of these programs was solved using CBC with a predefined time limit.

In our experiments we applied three different time limits. 1 minute per vertex, 10 minutes per vertex and 1 hour per vertex were used. For each of the integer programs with the fixed initial vertex, the ILP solver can reach the optimum or obtain a timeout and a possibly suboptimal solution. If the number of vertices which can still potentially form a cycle is not higher than the best subproblem solution found so far, then this solution can be deemed optimal. This allowed us to categorise the problem instances, based on the time limit that was sufficient to solve the instance based on our formulation.

Easy instances are those for which the optimum was found with 1 minute time limit per vertex. *Medium difficulty instances* were solved using 10 minute time limit per vertex. The rest of the graphs were classified as *hard instances.*

The data mining pipeline was implemented as a Python script. This script manages both the initial vertex choice and the ILP program generation. This involves launching CBC and maintaining whether the optimum has been found or another integer program has to be generated.

The heuristic approach was implemented in C++ using the Qt toolkit. Similarly to the exact approach, the heuristic was also applied to the graphs obtained by iterative pruning of the leaves.

All experiments were run on an Apple Mac Pro running OS X El Capitan with a 3.5GHz 6-Core Intel Xeon E5 CPU and with 16 GB 1866 MHz DDR3 RAM.

It is worth noting that pruning of the leaves can significantly reduce the instance size. For example, the network *gplus*_200 of 200 vertices representing the public circles data from Google+ was reduced from 200 to 118 vertices. This improves the performance of both of the exact approaches and the heuristic.

For the experimental evaluation, we have used a wide range of real-world networks of small to medium sizes. The data set represents networks from four different sources and includes social networks, a research collaboration network, a neural network, coappearance networks for literary classics and protein-protein interaction networks.

These included a small 52-vertex sample of a social network, as well as larger samples from social networks Google+ and Pokec which is a Slovak social network [39]. A large-scale snapshot of this network is a part of the SNAP network data set [29]. Graphs from Newman's network data repository come from diverse domains. Network *adjnoun* represents adjective-noun adjacencies for David Copperfield, *football* is an American college football network and *lesmis* is a coappearance network for Les Miserables. Other networks include *netscience* which represents network science collaborations, *zachary* is a social network of a karate club, *celegansneural* is a neural network for nematode worm Caenorhabditis elegans, *dolphins* is a social network of bottlenose dolphins and *polbooks* represents a network of books about US politics.

The next subset of instances are the protein-protein interaction networks from the UCLA database of interacting proteins [38, 47, 48, 49]. These networks can represent the full data, "core" data (this is indicated by the *CR* suffix) or high-throughput data (which is indicated by the *HT* suffix). *Celeg20160114CR* represents a protein-protein interaction network for Caenorhabditis elegans. *Dmela20160114CR* is a network for Drosophila melanogaster, the fruit fly. *Ecoli20160114CR* represents the protein-protein interactions for Escherichia coli. *Hpylo20160114* is the network for Helicobacter pylori, a bacterium associated with chronic gastritis. *Hsapi20160114HT* is a high-throughput human protein-protein interaction network. Last but not least, *Mmusc20160114CR* is a protein-protein interaction network for the house mouse.

The last group represents the coappearance networks for several literary classics from the DIMACS graphs [24]. Instance *anna* represents the network for Anna Karenina, *david* is the coappearance network for David Copperfield, *homer* is the network for Iliad and Odyssey and *huck* represents the network for Huckleberry Finn.

4.2. Numerical Results

In our experiments, we embedded both the ILP approach based on our formulation and Dixon and Goodman's formulation into the data mining pipeline. Both formulations represent the problem of finding the longest simple cycle with fixed initial vertex. We have used experiments with time limit of 1 minute per vertex to compare the efficiencies of the approaches.

These results are summarised in Table 1. The rows represent the networks in which we searched for the longest cycle. The columns represent the length of the longest cycle found and time taken in seconds to find this cycle by our approach (our ILP formulation column) and the previous approach (Dixon and Goodman's ILP formulation column). If the column named optimum contains a numerical value, it is the proven optimum. If the cycle length starts with the $\geq$ sign, then the value represents a lower bound of the longest cycle length, i.e. the cycle has not been proven to be the longest within the time limit. If N/A is included instead of a numerical value, then no cycle has been found within the time limit.

These results show that the approach based on our formulation significantly outperforms the equivalent approach based on the previous formulation. Our formulation led to 11 out of 22 instances being solved within this time limit. Whereas, the previous formulation found the provable optima only for 3 instances. The time taken to find these solutions also indicates that the approach based on our formulation takes considerably less computational time to produce equivalent solutions. This is most probably due to the number and the character of our flow-based constraints, which seem to lead to a less constrained search space. We have also performed equivalent experiments with 10 minute time limit, which we omit due to space limitations. It is worth noting that the previous formulation led to 5 out of 22 solutions being found. This is still considerably less successful than our approach with 1 minute time limit.

Table 2 presents the results we obtained using our exact approach. The columns represent the three strategies with 1 minute, 10 minute and 1 hour time limit. The meanings of numerical values and symbols in the table are equivalent to their respective meanings in Table 1.

We used these experiments with our formulation to categorise the instances into easy, medium and hard difficulty. The results in Table 2 indicate that for the 22 networks we studied, 11 instances were easy, 3 were of a medium difficulty and 8 instances remained hard. If an instance has been solved within a shorter time limit, we omit the experiments with higher time limits.

Easy instances included *soc_52*, *adjnoun*, *football*, *lesmis*, *zachary*, *dolphins*, *polbooks*, *Celeg20160114CR*, *Dmela20160114CR*, *Hsapi20160114HT* and *huck*. For these instances the longest cycle was found in less than an hour. All four categories have a representative between the easy instances. There seems to be no correlation between the application domain from which the instance comes and the difficulty of the instance.

Medium difficulty instances can be solved optimally with a less restrictive time limit of 10 minutes per vertex. These include *gplus_200*, *celegansneural* and *david*.

Instances categorised as hard could not be solved with any time limit. These include *gplus_500*, *pokec_500*, *netscience*, *Ecoli20160114CR*, *Hpylo20160114*, *Mmusc20160114*, *anna* and *homer*.

It is worth noting that the structure of the graph seems to be the main factor that influences the difficulty. For example *anna* seems to be a hard instance despite its relatively low number of vertices. The social networks with 500 vertices were categorised to hard instances while the social network with 200 vertices has fallen into the medium difficulty. This indicates that the size of the graph is determining the difficulty but is not the only factor.

Summarising the results of our exact approach, we have found the optimal solutions for 14 out of 22 instances.

In Table 3, we include the results of the heuristic approach. The variants of the heuristic we have used include a version with perturbation operators 1, 2 and 3 (MSLS-10000-III) and a version with all 4 perturbation operators (MSLS-10000-IV). A multi-start version of the heuristic was used with 10000 consecutive runs. This is due to the fact that perturbation operator 4 helps to provide better results for some instances. However, for a majority of the instances the variant with the 3 operators was more efficient. An extended variant with 100000 runs was also used. Based on the results from 10000 runs, only the perturbation operators 1, 2 and 3 were used in this variant (MSLS-100000-III).

The heuristic was able to find the optimal solutions for most of the easy instances. For 8 out of 11 easy instances, both MSLS-10000-III and MSLS-10000-IV were able to find the optimum. For *Hsapi20160114HT*, MSLS-100000-III was able to find the longest cycle. This is the cycle depicted in Figure 1. For *adjnoun* and *polbooks*, only suboptimal solutions were found by the heuristic approach.

For the medium difficulty instances, the heuristic approach produces solutions with varied quality. For *gplus_200* and *david*, the obtained results are very close to the optimum. However, the longest cycle found for *celegansneural* is still 9 vertices shorter than the optimum.

For the hard instances, the exact approach has only found suboptimal solutions. Apart from *netscience*, these solutions are consistently better than the solutions found by the heuristic. However, the gap between the length of a cycle found by the exact approach and the heuristic varies between

Table 1: Computational results of the exact approach based on our ILP formulation and CBC (on the left-hand side), compared to the results of an exact approach based on the original problem formulation [14]. Both approaches were used with a time limit of 1 minute per vertex. N/A indicates that no cycle has been found within the time limit. These results indicate that the approach based on our formulation clearly outperforms the approach based on the original formulation.

Graph	CBC (1 min / vertex) our ILP Formulation		CBC (1 min / vertex) Dixon and Goodman's ILP Formulation	
	optimum	time	optimum	time
Social networks				
gplus_200	≥ 70	5117 s	≥ 60	6887 s
gplus_500	≥ 8	16353 s	N/A	18375 s
pokec_500	≥ 127	17485 s	N/A	20005 s
soc_52	51	15 s	51	2034 s
Graphs from Newman's network data repository				
adjnoun	101	207 s	≥ 100	6063 s
football	115	4 s	≥ 113	7147 s
lesmis	49	490 s	≥ 49	3683 s
netscience	≥ 88	25495 s	N/A	66573 s
zachary	20	19 s	≥ 20	1616 s
celegansneural	≥ 279	17959 s	N/A	17150 s
dolphins	53	≤ 1 s	53	22 s
polbooks	105	60 s	≥ 103	6003 s
Protein-protein interactions from UCLA database of interacting proteins				
Celeg20160114CR	6	≤ 1 s	6	199 s
Dmela20160114CR	14	49 s	N/A	5342 s
Ecoli20160114CR	≥ 8	18380 s	N/A	21695 s
Hpylo20160114	N/A	26484 s	N/A	25364 s
Hsapi20160114HT	64	1959 s	≥ 61	5167 s
Mmusc20160114CR	≥ 170	22012 s	N/A	32981 s
DIMACS graphs				
anna	≥ 77	6490 s	≥ 75	6522 s
david	≥ 72	4621 s	≥ 66	4685 s
homer	≥ 114	18309 s	N/A	19412 s
huck	48	858 s	N/A	3734 s

Table 2: Detailed computational results of the exact approach based on our ILP formulation and CBC. Three versions of the approach were used with time limits of 1 minute, 10 minutes and 1 hour per vertex for each instance. Whenever an instance was successfully solved with a lower time limit, the experiments with a higher time limit have been omitted. This strategy allowed us to categorise the instances to easy, medium and hard. Such a categorisation is based on whether the more restrictive versions of the exact approach were successful in finding the proven optimum or not. N/A indicates that no cycle has been found within the time limit.

graph	CBC (1 min / vertex)		CBC (10 min / vertex)		CBC (1 hour / vertex)	
	optimum	time	optimum	time	optimum	time
Social networks						
gplus_200	≥ 70	5117 s	70	30850 s		
gplus_500	≥ 8	16353 s	≥ 202	157858 s	≥ 206	942816 s
pokec_500	≥ 127	17485 s	≥ 163	168396 s	≥ 166	1003664 s
soc_52	51	15 s				
Graphs from Newman's network data repository						
adjnoun [35]	101	207 s				
football [19]	115	4 s				
lesmis [27]	49	490 s				
netscience [35]	≥ 88	25495 s	≥ 104	194185 s	≥ 107	868606 s
zachary [51]	20	19 s				
celegansneural [46]	≥ 279	17959 s	280	2785 s		
dolphins [32]	53	≤ 1 s				
*polbooks**	105	60 s				
Protein-protein interactions from UCLA database of interacting proteins [38, 49, 47, 48]						
Celeg20160114CR	6	≤ 1 s				
Dmela20160114CR	14	49 s				
Ecoli20160114CR	≥ 8	18380 s	≥ 242	179389 s	≥ 244	1068437 s
Hpylo20160114	N/A	26484 s	≥ 291	260322 s	≥ 299	1551768 s
Hsapi20160114HT	64	1959 s				
Mmusc20160114CR	≥ 170	22012 s	≥ 256	215912 s	≥ 267	1282218 s
DIMACS graphs [24]						
anna	≥ 77	6490 s	≥ 79	64498 s	≥ 79	387458 s
david	≥ 72	4621 s	72	4714 s		
homer	≥ 114	18309 s	≥ 223	171157 s	≥ 234	1015438 s
huck	48	858 s				

* Network *polbooks* has not been published in a past paper. It is available from Newman's network data repository: `http://www-personal.umich.edu/∼mejn/netdata/`.

Table 3: Computational results of the multi-start local search heuristics with 10000 trials with 3 or 4 perturbation operators (MSLS-10000-III and MSLS-10000-IV) and 100000 trials with 3 perturbation operators (MSLS-100000-III). Their comparison to the proven longest cycle lengths (or their lower bounds) found by the exact approach based on CBC is included.

graph	CBC	MSLS-10000-III		MSLS-10000-IV		MSLS-100000-III	
		cycle length	time	cycle length	time	cycle length	time
				Social networks			
gplus_200	70	67	65 s	66	50 s	68	729 s
gplus_500	$\geq$ 206	186	606 s	186	560 s	192	6891 s
pokec_500	$\geq$ 166	155	663 s	151	598 s	159	7158 s
soc_52	51	51	25 s	51	23 s	51	280 s
				Graphs from Newman's network data repository			
adjnoun [35]	101	91	78 s	92	62 s	93	967 s
football [19]	115	115	103 s	115	93 s	115	1246 s
lesmis [27]	49	49	18 s	49	14 s	49	204 s
netscience [35]	$\geq$ 107	107	531 s	107	524 s	108	5944 s
zachary [51]	20	20	4 s	20	2 s	20	52 s
celegansneural [46]	280	270	2147 s	267	1960 s	271	27953 s
dolphins [32]	53	53	7 s	53	7 s	53	94 s
*polbooks**	105	104	73 s	103	65 s	104	802 s
Protein-protein interactions from UCLA database of interacting proteins [38, 49, 47, 48]							
Celeg20160114CR	6	6	1 s	6	6 s	6	5 s
Dmela20160114CR	14	14	3 s	14	3 s	14	43 s
Ecoli20160114CR	$\geq$ 244	207	894 s	205	849 s	211	11208 s
Hpylo20160114	$\geq$ 299	239	1818 s	235	1674 s	241	21357 s
Hsapi20160114HT	64	63	22 s	63	17 s	64	253 s
Mmusc20160114CR	$\geq$ 267	243	728 s	248	712 s	246	8353 s
				DIMACS graphs [24]			
anna	$\geq$ 79	76	96 s	77	112 s	77	1316 s
david	72	71	50 s	70	45 s	71	608 s
homer	$\geq$ 234	206	1508 s	205	1468 s	209	17981 s
huck	48	48	24 s	48	21 s	48	284 s

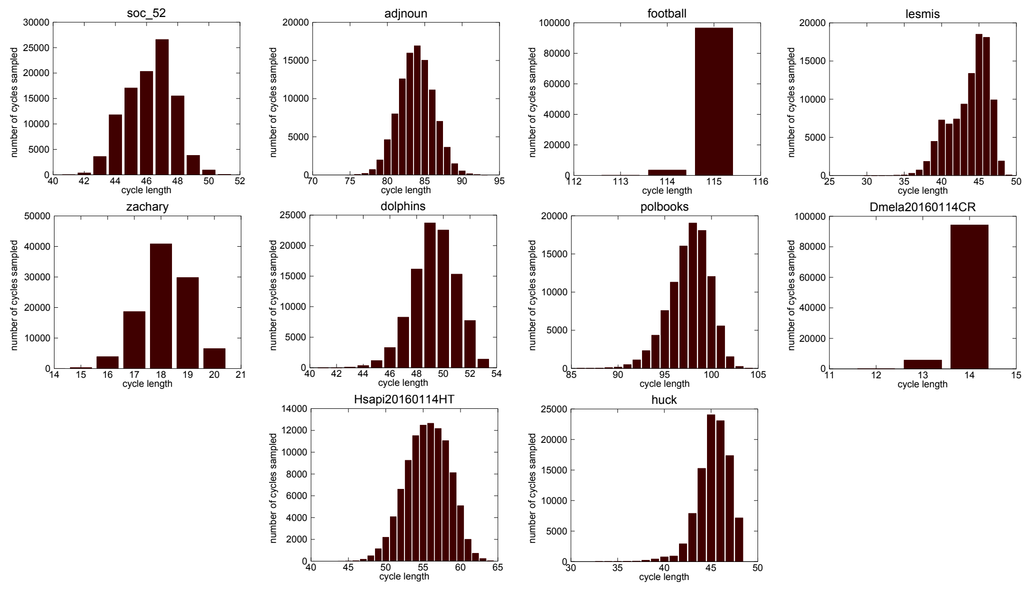

Figure 4: Distributions of cycle lengths sampled by the multi-start local search approach MSLS-100000-III for the easy instances.

instances. For *anna*, the exact approach found a suboptimum on 79 vertices while the heuristic found a cycle on 77 vertices. However, for *Hpylo20160114* the exact approach found a suboptimum on 299 vertices while the result of the heuristic was more distant with 241 vertices.

The instance *netscience* is particularly interesting, since the heuristic found a cycle of length 108. On the other hand, the exact approach with 1 hour time limit per vertex only found a cycle of length 107. This is somewhat surprising and seems to be related to the structure of this graph. Note that MSLS-10000-III and MSLS-10000-IV were able to find a cycle on 107 vertices in minutes, while the exact approach needed more than a week to produce such a cycle. This indicates that although the exact approach usually provides better results, the heuristic can perform surprisingly well for certain instances.

5. Discussion

The heuristic approach performs a sequence of samplings of long simple cycles using DFS and their improvement using a local search approach with

Table 4: The statistical properties of cycle length distributions obtained by MSLS-100000-III. This includes the longest cycle length found, the average cycle length μ, the standard deviation σ, the skewness γ_1 and the "excess" kurtosis γ_2 for each distribution.

graph	optimum	MSLS-100000-III				
		best	μ	σ	γ_1	γ_2
	Social networks					
gplus_200	70	68	59.46	2.8421	-0.42985	-0.040028
gplus_500	$\geq$ 206	192	161.33	7.2337	-0.031828	-0.083499
pokec_500	$\geq$ 166	159	130.83	8.1637	-0.41967	-0.058592
soc_52	51	51	46.202	1.5434	-0.15057	-0.44118
	Graphs from Newman's network data repository					
adjnoun [35]	101	93	83.852	2.3339	0.0088051	-0.066728
football [19]	115	115	114.96	0.19147	-5.313	28.502
lesmis [27]	49	49	43.772	2.5782	-0.67178	-0.27657
netscience [35]	$\geq$ 108	108	92.438	5.4751	-0.21830	-0.25129
zachary [51]	20	20	18.684	0.74029	-0.15538	-0.053173
celegansneural [46]	280	271	260.33	2.884	-0.11364	0.005146
dolphins [32]	53	53	49.323	1.6584	-0.30721	0.081956
polbooks	105	104	97.632	2.1877	-0.54051	-0.058592
Protein-protein interactions from UCLA database of interacting proteins [38, 49, 47, 48]						
Celeg20160114CR	6	6	6	0	N/A	N/A
Dmela20160114CR	14	14	13.943	0.23244	-3.8124	12.548
Ecoli20160114CR	$\geq$ 244	211	183.21	6.1459	-0.01982	-0.013013
Hpylo20160114	$\geq$ 299	241	214.22	5.2453	0.23223	0.10657
Hsapi20160114HT	64	64	55.581	2.8587	-0.17028	-0.30979
Mmusc20160114CR	$\geq$ 267	246	207.74	9.4577	0.077919	-0.074851
		DIMACS graphs [24]				
anna	$\geq$ 79	77	67.081	3.1581	-0.41567	0.243
david	72	71	63.918	2.5469	-0.61604	0.54436
homer	$\geq$ 234	209	183.03	6.457	-0.045365	-0.0058615
huck	48	48	45.273	1.6975	-0.85419	1.6703

our perturbation operators. Therefore, we are interested in the distribution of cycle lengths sampled in different runs. We have explored these distributions for MSLS-100000-III.

Figure 4 illustrates the distributions we obtained for the easy instances. For most of the instances, the peaks of the distributions are located near the far end of the distributions. The probability of sampling a cycle which is longer than the peak length seems to decline quite rapidly. Therefore, we have decided to measure the statistical properties of these distributions. Apart from the mean and the standard deviation, these properties also include the skewness and the kurtosis.

Table 4 presents these properties for each distribution X obtained for each graph. The columns of the table represent the optimum, the best result obtained by MSLS-100000-III, the mean (μ), the standard deviation (σ),

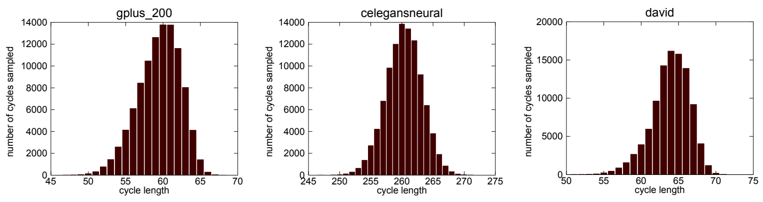

Figure 5: Distributions of cycle lengths sampled by the multi-start local search approach MSLS-100000-III for the medium difficulty instances.

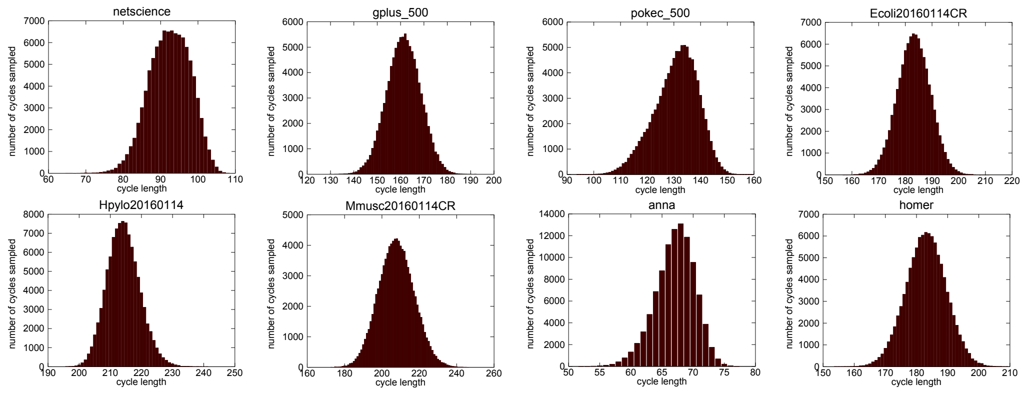

Figure 6: Distributions of cycle lengths sampled by the multi-start local search approach MSLS-100000-III for the hard instances.

the skewness ($\gamma_1 = E[(X-\mu)^3]/\sigma^3$) and the "excess" kurtosis ($\gamma_2 = E[(X-\mu)^4]/\sigma^4 - 3$). Based on these results, a distribution with a low value of γ_1 and a high value of γ_2 seems to indicate an easy instance. On the other hand, the distribution for *adjnoun* has both γ_1 and γ_2 close to 0. Since *adjnoun* was the only easy instance for which MSLS-100000-III was relatively far away from the optimum, a distribution close to a normal distribution may indicate that the heuristic is less efficient. It is also worth noting that apart from *adjnoun*, $\gamma_1 < 0$ for all easy instances.

On a related note, the distributions seem to differ from instance to instance. The profile for *polbooks* suggests that sampling of the longest cycle may be possible with an increased number of runs.

Figure 5 illustrates the distributions we obtained for the medium difficulty instances. Compared to the easy instances, the plots for medium difficulty indicate more fine-grained distributions. For all of these instances we have that $\gamma_1 < 0$. However the distribution for *celegansneural* has γ_1 and γ_2 close to 0. Similarly to *adjnoun*, the heuristic has not managed to find the optimum for this instance. This indicates that a more sophisticated approach may be more suitable for this type of difficulty. Hybrid heuristics such as evolutionary algorithms or algorithms based on swarm intelligence may be the suitable tools to overcome the local optima for these instances. It is also worth noting that the distributions obtained by different heuristics can be used as a promising tool to compare the efficiency of different approaches.

Such algorithms could also be beneficial for the hard instances. The distributions are depicted in Figure 6. Apart from the instance *anna*, the distributions seem to have similar fine-grained profiles. For most of these instances the heuristic has found cycles with moderate length. The results of the exact approach indicate that providing significantly longer cycles seems to require quite extensive computational efforts.

It is worth noting that the perturbation operators contributed to these values relatively significantly. DFS used as an initialisation procedure for our heuristic provides moderately long cycles. The perturbation operators 1 and 2 use the typical properties of real-world networks to enlarge such a cycle. While the perturbation operator 1 seems to work well for graphs with significant community structure, perturbation operator 2 seems to be suitable for graphs where the underlying structure is a grid. The role of the perturbation operators 3 and 4 was in searching in the areas of the search space with equally long cycles. For a majority of the instances, perturbation operator 3 was found to be the more efficient plateau exploration operator.

This work can also be used as a basis for more application oriented studies. Topics such as layout algorithms for social and biological networks [5, 19], drawing of planar graphs [40] or the closely related longest path problem [25, 42] can be the avenues to explore.

6. Conclusions

We have proposed two approaches to find long simple cycles in real-world complex networks. The first technique is an exact approach based on our integer linear programming (ILP) formulation of the problem. The exact approach utilises a data mining pipeline which solves the longest simple cycle problem as a sequence of these ILP programs for different fixed initial vertices. The experimental results have shown that our approach considerably outperforms an equivalent approach based on a previous ILP formulation of the problem. Our approach can also be used to find the provably longest cycle for graphs of small to medium size.

The second technique is a heuristic which combines an initial construction of a long simple cycle using depth-first search with several perturbation operators used to improve this initial solution. Multi-start versions of this heuristic provide relatively long cycles in a limited time. For easier and medium difficulty instances the heuristic provided optimal or near-optimal solutions to the problem. For very large instances the exact approach becomes unscalable, possibly requiring weeks or even months of computation. For these instances the heuristic approach is much more valuable for real-world applications.

We believe that our study can form a basis for further explorations of the problem in real-world scenarios. The applications in social and biological networks represent one such route. For large scale instances of the problem new heuristics combining different ideas and operators may contribute to develop more scalable approaches suitable for finding even longer cycles.

In summary this paper has laid down the fundamental ideas for practical algorithms which can be used to find long cycles in real-world networks.

References

[1] R. Albert and A. L. Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[2] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11–24, 1989.

[3] M. Aung. Longest cycles in triangle-free graphs. *Journal of Combinatorial Theory, Series B*, 47(2):171–186, 1989.

[4] A. L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[5] M. Y. Becker and I. Rojas. A graph layout algorithm for drawing metabolic pathways. *Bioinformatics*, 17(5):461–467, 2001.

[6] G. Bianconi and M. Marsili. Loops of any size and Hamilton cycles in random scale-free networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(06):P06005, 2005.

[7] A. Björklund and T. Husfeldt. Finding a path of superlogarithmic length. *SIAM Journal on Computing*, 32(6):1395–1402, 2003.

[8] A. Björklund, T. Husfeldt, and S. Khanna. Approximating longest directed paths and cycles. In *Automata, Languages and Programming*, pages 222–233. Springer, 2004.

[9] H. L. Bodlaender. On linear time minor tests with depth-first search. *Journal of Algorithms*, 14(1):1–23, 1993.

[10] P. Bonami, L. T. Biegler, A. R. Conn, G. Cornuéjols, I. E. Grossmann, J. Laird, C. D.and Lee, A. Lodi, F. Margot, N. Sawaya, et al. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5(2):186–204, 2008.

[11] F. Chen and K. Li. Detecting hierarchical structure of community members in social networks. *Knowledge-Based Systems*, 87:3–15, 2015.

[12] G. Chen, Z. Gao, X. Yu, and W. Zang. Approximating the longest cycle problem on graphs with bounded degree. In *Computing and Combinatorics*, pages 870–884. Springer, 2005.

[13] J. Cohen. Bioinformatics - an introduction for computer scientists. *ACM Computing Surveys*, 36(2):122–158, 2004.

[14] E. T. Dixon and S. E. Goodman. An algorithm for the longest cycle problem. *Networks*, 6(2):139–149, 1976.

[15] V. Ejov, J. Filar, and J. Gondzio. An interior point heuristic for the Hamiltonian cycle problem via Markov decision processes. *Journal of Global Optimization*, 29(3):315–334, 2004.

[16] T. Feder, R. Motwani, and C. Subi. Approximating the longest cycle problem in sparse graphs. *SIAM Journal on Computing*, 31(5):1596–1607, 2002.

[17] H. N. Gabow. Finding paths and cycles of superpolylogarithmic length. *SIAM Journal on Computing*, 36(6):1648–1671, 2007.

[18] B. Gavish and S. C. Graves. The traveling salesman problem and related problems, 1978.

[19] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[20] J. L. Gross, J. Yellen, and P. Zhang. *Handbook of graph theory.* CRC press, 2014.

[21] K. A. Hawick and H. A. James. Enumerating circuits and loops in graphs with self-arcs and multiple-arcs. In *Proc. 2008 Int. Conf. on Foundations of Computer Science (FCS08)*, pages 14–20, Las Vegas, USA, 14-17 July 2008. CSREA.

[22] K.A. Hawick, H.A. James, and C.J. Scogings. Structural Circuits and Attractors in Kauffman Networks. In Hussein A. Abbass and Marcus Randall, editors, *Proc. Third Australian Conference on Artificial Life*, volume 4828 of *LNCS*, pages 189–200, Gold Coast, Australia, 4-6 December 2007. Springer. 978-3-540-76930-9.

[23] D. B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84, 1975.

[24] D. S. Johnson and M. Trick. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge.* American Mathematical Society, Providence, RI, 1996.

[25] D. Karger, R. Motwani, and G. D. S. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18(1):82–98, 1997.

[26] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.

[27] D. E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing.* Addison-Wesley, Reading, MA, 1993.

[28] A. Leist and K. A. Hawick. Circuits as a classifier for small-world network models. In *Proc. WORLDCOMP 2009 International Conference on Foundations of Computer Science (FSC 09) Las Vegas, USA*, number CSTN-003, 13-16 July 2009.

[29] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[30] J. T. Linderoth and A. Lodi. MILP software. *Wiley encyclopedia of operations research and management science*, 2011.

[31] K. Liu, J. Huang, H. Sun, M. Wan, Y. Qi, and H. Li. Label propagation based evolutionary clustering for detecting overlapping and non-overlapping communities in dynamic networks. *Knowledge-Based Systems*, 89:487–496, 2015.

[32] D. Lusseau, K. Schneider, O. J. Boisse, P. Haase, E. Slooten, and S. M. Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, 54(4):396–405, 2003.

[33] E. Marinari, G. Semerjian, and V. Van Kerrebroeck. Finding long cycles in graphs. *Physical Review E*, 75(6):066708, 2007.

[34] B. Monien. How to find long paths efficiently. *North-Holland Mathematics Studies*, 109:239–254, 1985.

[35] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74(036104):036104–1–036104–19, 2006.

[36] T. Nishizeki, T. Asano, and T. Watanabe. An approximation algorithm for the hamiltonian walk problem on maximal planar graphs. *Discrete Applied Mathematics*, 5(2):211–222, 1983.

[37] G. Pandurangan. On a simple randomized algorithm for finding long cycles in sparse graphs, 2004.

[38] L. Salwinski, C. S. Miller, A. J. Smith, F. K. Pettit, J. U. Bowie, and D. Eisenberg. The database of interacting proteins: 2004 update. *Nucleic acids research*, 32(suppl 1):D449–D451, 2004.

[39] L. Takac and M. Zabovsky. Data analysis in public social networks. In *International Scientific Conference and International Workshop Present Day Trends of Innovations*, pages 1–6, 2012.

[40] R. Tamassia. *Handbook of graph drawing and visualization*. CRC press, 2013.

[41] R. Tarjan. Enumeration of the elementary circuits of a directed graph. *SIAM Journal on Computing*, 2(3):211–216, 1973.

[42] R. Uehara and Y. Uno. Efficient algorithms for the longest path problem. In *Algorithms and computation*, pages 871–883. Springer, 2005.

[43] R. Uehara and Y. Uno. On computing longest paths in small graph classes. *International Journal of Foundations of Computer Science*, 18(05):911–930, 2007.

[44] I. Wagner, A. M. Bruckstein, et al. Hamiltonian (t)-an ant-inspired heuristic for recognizing Hamiltonian graphs. In *Proceedings of the 1999 Congress on Evolutionary Computation, 1999. CEC 99*, volume 2. IEEE, 1999.

[45] D. J. Watts. *Small worlds: the dynamics of networks between order and randomness*. Princeton university press, 1999.

[46] D. J. Watts and S. H. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393(6684):440–442, 1998.

[47] I. Xenarios, E. Fernandez, L. Salwinski, X. J. Duan, M. J. Thompson, E. M. Marcotte, and D. Eisenberg. Dip: the database of interacting proteins: 2001 update. *Nucleic acids research*, 29(1):239–241, 2001.

[48] I. Xenarios, D. W. Rice, L. Salwinski, M. K. Baron, E. M. Marcotte, and D. Eisenberg. Dip: the database of interacting proteins. *Nucleic acids research*, 28(1):289–291, 2000.

[49] I. Xenarios, L. Salwinski, X. J. Duan, P. Higney, S. M. Kim, and D. Eisenberg. Dip, the database of interacting proteins: a research tool for studying cellular networks of protein interactions. *Nucleic acids research*, 30(1):303–305, 2002.

[50] F. Yang, R. Zhang, Y. Yao, and Y. Yuan. Locating the propagation source on complex networks with propagation centrality algorithm. *Knowledge-Based Systems*, 100:112–123, 2016.

[51] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.