Polynomial algorithms for computing the isolated toughness of interval and split graphs

Fengwei Li1  |  Qingfang Ye1  |  Hajo Broersma2  |  Xiaoyan Zhang3

1College of Basic Science, Ningbo University of Finance and Economics, Ningbo, PR. China
2Faculty of EEMCS, University of Twente, Enschede, the Netherlands
3School of Mathematical Sciences and Institute of Mathematics, Nanjing Normal University, Nanjing, PR. China

Correspondence
Xiaoyan Zhang, School of Mathematical Sciences and Institute of Mathematics, Nanjing Normal University, Nanjing, Jiangsu 210023, PR. China.
Email: zhangxiaoyan@njnu.edu.cn

Abstract
The isolated toughness $ir(G)$ of a noncomplete graph $G$ is defined as: $ir(G) = \min \{\frac{|Y|}{\delta_G(V(G) \setminus Y)} : Y \subseteq V(G), \delta_G(V(G) \setminus Y) > 1\}$, where $C(G)$ is the collection of all vertex cutsets of $G$ and $|i(G) - Y|$ stands for the number of isolated vertices in $G - Y$. If $G$ is a complete graph, we set $ir(G) = \infty$. This isolated toughness parameter is closely related to the existence of factors and fractional factors in graphs. These factors and fractional factors are well-studied within graph theory, and have various applications in several fields related to computer science. In this article, we pay our attention to the computational complexity of computing the isolated toughness. We present polynomial algorithms for computing the exact value of $ir(G)$ for interval graphs and for split graphs, two well-studied special graph classes.

KEYWORDS
factor, fractional factor, interval graph, isolated toughness, polynomial time algorithm, split graph

1  |  INTRODUCTION

The research reported here is strongly motivated by the key role the graph parameter isolated toughness plays in the theory regarding the existence of factors and fractional factors in graphs. Before we move on to explaining the relevance of this isolated toughness concept, we cannot avoid the introduction of some terminology and notation. Interested readers who are familiar with the terminology and notation involved in this kind of research might want to skip this part and start reading from definition 1.1.

For readers that are less familiar with graph theoretic notions, we start this introduction by recalling some basic graph theoretic definitions and notations that are used throughout the article. Our definitions and notations are mainly based on and adopted from the textbook due to Bondy and Murty.

Throughout the article, we only consider undirected finite graphs $G = (V(G), E(G))$ containing no loops or multiple edges. For a nonempty subset $S \subseteq V(G)$, we use $G[S]$ to stand for the induced subgraph of $G$ by $S$. The degree of a vertex $v$ in $G$ is denoted by $d_G(v)$, or simply $d(v)$ if $G$ is clear from the context. We use $\delta(G) = \min_{v \in V(G)}\{d(v)\}$ to denote the minimum degree of the graph $G$. With the open neighborhood of any $v \in V(G)$ we mean the set $N(v) = \{w \in V(G) : (v, w) \in E(G)\}$. Analogously, for any nonempty set $S \subseteq V(G)$, $N(S) = \bigcup_{u \in S}N(u)$ expresses the open neighborhood of $S$. As usual, $\kappa(G)$ stands for the connectivity of $G$. An independent set of $G$ is a nonempty set $S \subseteq V(G)$ so that the induced subgraph $G[S]$ is edgeless. With $n$-vertex graph we indicate a graph on $n$ vertices. Furthermore, with $(n, m)$-graph we denote a graph $G$ having order $n = |V(G)|$ and size $m = |E(G)|$. For any proper subset $S \subseteq V(G)$, we let $\omega(G - S)$ stand for the number of (connected) components in $G - S$.

A subset $S \subseteq V(G)$ is a (vertex) cutset in $G$ if $\omega(G - S) \geq 2$. We call $G[S]$ a clique of $G$ if it is a complete induced subgraph. $C(G)$ denotes the collection of all cutsets in $G$. Note that $S \subseteq C(G)$ iff $G$ is not connected. We denote $y \sim e$ if $y \in V(G)$ is an end vertex of $e \in E(G)$. 
1.1 Factors and fractional factors

The systematic study of factors in graphs is a rather classical topic within graph theory, but the analogous study of fractional factors of graphs is a relatively new problem area in graph theory, dating back to the 1990s. For two nonnegative integer-valued functions \( \varphi \) and \( \psi \) with domain \( V(G) \) meeting the conditions \( \varphi(y) \leq \psi(y) \) for all \( y \in V(G) \), the spanning subgraph \( F \) of \( G \) with the property that \( \varphi(y) \leq d_F(y) \leq \psi(y) \) holds for each \( y \in V(G) \) is called a \((\varphi, \psi)\)-factor of \( G \). Similarly, \( F \) is called a \((\varphi, \psi)\)-factor of \( G \) if \( \varphi(y) = d_F(y) = \psi(y) \) for all \( y \in V(G) \). Let \( c < d \) be two positive integers. For any \( y \in V(G) \), if \( \varphi(y) \equiv c \) and \( \psi(y) \equiv d \), a \((\varphi, \psi)\)-factor is also called a \([c, d]\)-factor. Moreover, if \( \varphi(y) = \psi(y) = s \) for all \( y \in V(G) \), a \((\varphi, \psi)\)-factor becomes what is usually called an \( s\)-factor of \( G \).

By considering a broader \([0, 1]\)-valued indicator function instead of an integer-valued function, one can generalize the above traditional factors to fractional factors. For this purpose, we define a real-valued function \( \varphi \) on \( V(G) \), and let \( E_1 = \{e \in E(G) : \varphi(e) > 0 \} \). If \( \varphi(e) = s \) for all \( e \in V(G) \), then the spanning subgraph \( G(E) \) is said to be a fractional \( s\)-factor of \( G \). A fractional \( 1\)-factor is also called a fractional perfect matching. If inequality \( \varphi(y) \leq \sum_{e \in \{y\}} \varphi(e) \) is satisfied for all \( y \in V(G) \), then the graph \( F = (V(G), E) \) is a so-called fractional \((\varphi, \psi)\)-factor of \( G \) with indicator function \( \varphi \). If \( \varphi(e) \in [0, 1] \) for all \( e \in E(G) \), then \( F \) is a \((\varphi, \psi)\)-factor of \( G \).

Fractional factors have appeared in a variety of application domains, including network design and network scheduling. For example, in a communication network, if one has to transmit large packets of data through the network from several senders to several receivers by means of multiple routes, the network’s productiveness (throughput) can usually be improved if the large packets are separated into smaller parts. In such cases one can use a fractional flow problem to model and simulate the practicable arrangement of data packets to channels. When the destinations and sources of the network are nonintersecting, the problem changes into a fractional matching problem.

1.2 Toughness and isolated toughness

In a publication dating back to 1973, Chvátal introduced and studied the toughness of a graph, and recommended to consider this new graph parameter in the study of regular factors of graphs. In Reference 3, the toughness \( \tau(G) \) of a connected noncomplete graph \( G \) is defined as

\[
\tau(G) = \min \left\{ \frac{|Y|}{\omega(G-Y)} : Y \in C(G), \omega(G-Y) > 1 \right\}.
\]

Toughness, just like connectivity, is a useful graph vulnerability parameter, but toughness can also be used to study many basic properties of graphs, in particular properties related to the cycle and path structure of graphs. Interestingly, several of the conjectures that appeared in Reference 3 are still open, whereas others have been proved or refuted. With regard to factors in graphs, Chvátal conjectured that if \( \tau(G) \geq s \) for an integer \( s \geq 2 \), then \( G \) possesses an \( s\)-factor. About 10 years after the appearance of Reference 3, this conjecture was verified by Enomoto et al., using a classical factor result due to Tutte.

Inspired by Chvátal’s toughness concept, and by substituting \( \omega(G-Y) \) in place of \( \omega(G-Y) \) in the above definition of toughness, Ma and Liu introduced the so-called isolated toughness of a graph, denoted by \( ir(G) \). Here, \( \omega(G-Y) \) indicates the number of isolated vertices (i.e., vertices with degree 0) in \( G \). They mainly applied this newly introduced parameter as a measure to investigate conditions for a graph to have a (fractional) factor.

Definition [5] The isolated toughness of a connected noncomplete graph \( G \) is defined as

\[
ir(G) = \min \left\{ \frac{|Y|}{\omega(G-Y)} : Y \in C(G), \omega(G-Y) > 1 \right\}.
\]

where the minimum is obtained over all the cutsets of \( G \). For complete graphs \( K_n \), we simply define \( ir(K_n) = \infty \).

The following result is a basic result in fractional factor theory, and it implies a necessary and sufficient condition for the existence of a fractional \( 1\)-factor based on the isolated toughness.

**Theorem 1.** [2] There exists a fractional \( 1\)-factor in \( G \) if and only if \( |Y| \geq \omega(G-Y) \) for any \( Y \subseteq V(G) \).

Obviously, this result yields the following consequence.

**Theorem 2.** [6] Let \( G \) be an \( n\)-vertex graph with \( n \geq 2 \). Then there exists a fractional \( 1\)-factor in \( G \) if and only if \( ir(G) \geq 1 \).

A natural question is whether similar results can be obtained for the existence of fractional \( s\)-factors with \( s > 1 \). In the light of this question, Ma and Liu established that a graph \( G \) possesses a fractional \( 2\)-factor when \( ir(G) \geq 2 \) and \( \delta(G) \geq 2 \). Furthermore, more generally in Reference 7 they showed that a graph \( G \) contains a fractional \( s\)-factor if \( ir(G) \geq s \) and \( \delta(G) \geq s \). Moreover, there they showed the following: if \( \delta(G) \geq ir(G) \geq c-1 + \frac{c}{d} \), then \( G \) possesses a fractional \([c, d]\)-factor, where \( c < d \) are two positive integers. In another article, Ma and Yu proved that a graph \( G \) fulfilling \( \delta(G) \geq c \), \( ir(G) \geq c-1 + \frac{c}{d} \), and such that \( G - T \) has no \((c-1)\)-regular component for any \( T \subseteq V(G) \), has a fractional \([c, d]\)-factor. For more results on this topic.
we refer the interested reader to References 9, 10. It is clear from the above results that the isolated toughness plays a key role in the theory related to the existence of fractional factors.

If one wants to apply any of the above results involving conditions based on the isolated toughness, it is essential to obtain the exact value (or some reasonable lower bound on the value) of the isolated toughness of a given graph first. Motivated by this observation, and for this purpose, in our research we focused our attention on the complexity of calculating the exact value of the isolated toughness of a graph. In this article, we first design a polynomial algorithm for computing \( \tau(G) \) for interval graphs. Next, we prove that for split graphs determining \( \tau(G) \) is also polynomially solvable. We start the following section by recalling some additional terminology and stating some elementary results that we will use in our considerations. A preliminary version of this article appeared in the Proceeding of the 21th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2020).11

2 | PRELIMINARIES

In this section, we recall a number of notations, definitions and state some lemmas which we will use throughout the following sections. We first recall the definition of a minimal cutset as it was introduced in Reference 12.

**Definition** [12] Let \( c, d \in V(G) \) be two nonadjacent vertices. If for some subset \( S \subseteq V(G) \), \( c \) and \( d \) are in different components of \( G - S \), then \( S \) is called a \( c, d \)-cutset. If there does not exist a proper subset of \( S \) which is also a \( c, d \)-cutset of \( G \), then \( S \) is said to be a minimal \( c, d \)-cutset of \( G \). A minimal cutset \( S \subseteq V(G) \) is a minimal \( c, d \)-cutset for some nonadjacent vertices \( c \) and \( d \) in \( G \).

As a matter of convenience, let \( MC(G) \) denote the collection of all the minimal cutsets in \( G \). The below lemma supplies a simple way to check whether a given set \( S \subseteq V(G) \) is a minimal cutset in \( G \) or not.

**Lemma 1.** [13] A cutset \( S \in MC(G) \) if and only if \( G - S \) has at least two components so that any \( v \in S \) has a neighbor in each of the components.

**Definition** Let \( G \) be an \( n \)-vertex graph. Define \( i_k(G), k \in \{0, 1, 2 \ldots n\} \), as the maximum number of isolated vertices the graph \( G \) can obtain after accurately removing \( k \) vertices from \( G \), that is, \( i_k(G) = 0 \), and for \( k < n \)

\[
i_k(G) = \max\{i(G - S) : S \subseteq V, |S| = k\}.
\]

Using the above, one easily checks that for any noncomplete graph \( G \),

\[
\tau(G) = \min\{\frac{k}{i_k(G)} : i_k(G) > 1\}.
\]

We next give an expression to compute \( i_k(G) \) for \( k \in \{0, 1, 2 \ldots n\} \).

**Theorem 3.** Let \( G \) be an \( n \)-vertex noncomplete graph, and let \( k \in \{0, 1, 2 \ldots n\} \), if \( i_k(G) > 1 \), then

\[
i_k(G) = \max_{|S|<k} \max_{\sum_{j=1}^{q} i_j(G(C_j)) + p} \left\{ \sum_{j=p+1}^{q} r_j = k - |S^*| \right\}.
\]

where the maximum is obtained overall \( S^* \in MC(G) \), and over all nonnegative integer vectors \( (r_{p+1}, r_{p+2}, \ldots, r_q) \). Furthermore, \( C_1, C_2, \ldots, C_q \) are isolated vertices in \( G - S^* \), and \( C_{p+1}, C_{p+2}, \ldots, C_q \) are the components in \( G - S^* \) with \( |V(C_j)| \geq 2 \) for \( 0 \leq j \leq q \).

**Proof.** Suppose \( S \in C(G) \) is a cutset satisfying that \( |S| = k \) and \( i(G - S) = i_k(G) > 1 \). Let \( S^* \in MC(G) \) be a subset of \( S \), and assume \( C_1, C_2, \ldots, C_q \) are the isolated vertices in \( G - S^* \), and \( C_{p+1}, C_{p+2}, \ldots, C_q \) are the components in \( G - S^* \) with \( |V(C_j)| \geq 2 \) for \( 0 \leq j \leq q \). Hence \( C_1, C_2, \ldots, C_q \) must be components in \( G - S \), and we define \( S_j = S \cap C_j \) for \( 0 \leq j \leq q \). Then,

\[
i_k(G) = i_k(G - S) = \max_{|S|<k} \max_{\sum_{j=1}^{q} i_j(G(C_j)) + p} \left\{ \sum_{j=p+1}^{q} r_j = k - |S^*| \right\}.
\]

On the other hand, for \( S^* \in MC(G) \), let \( C_1, C_2, \ldots, C_q \) be the isolated vertices of \( G - S^* \), and let \( C_{p+1}, C_{p+2}, \ldots, C_q \) be the other components of \( G - S^* \). Let \( (r_{p+1}, r_{p+2}, \ldots, r_q) \) be a vector which maximizes the right-hand side of the aforementioned formula. For every \( j \in \{p + 1, p + 2, \ldots, q\} \), we choose a set \( S_j \) of \( G(C_j) \) so that \( |S_j| = r_j \) and \( i_j(G(C_j)) + p = i(G(C_j - S_j)) + p \). Hence, \( S = S^* \cup \bigcup_{j=p+1}^{q} S_j \) is a subset of \( V(G) \) and

\[
|S| = |S^*| + \sum_{j=p+1}^{q} |S_j| = |S^*| + \sum_{j=p+1}^{q} r_j = k.
\]
Furthermore, we have
\[
\max_{\mathcal{S} \subseteq \mathcal{S}'} \max_{S' \cup \mathcal{S} = \mathcal{S}'} \left\{ \sum_{j=p+1}^{q} \delta_j(G[S_j]) + p : \sum_{j=p+1}^{q} r_j = k - |S'| \right\}
\]
\[
= \sum_{j=p+1}^{q} \delta_j(G[S_j]) + p = \delta(G[S]) + p \leq \delta(G).
\]
This completes the proof.

**Theorem 4.** Let \( G \) be an \( n \)-vertex noncomplete graph, let \( S' \in \mathcal{S}(G) \), and let \( C_1, C_2, \ldots, C_p \) denote all isolated vertices in \( G - S' \), and \( C_{p+1}, C_{p+2}, \ldots, C_q \) all the components of \( G - S' \) such that \( |V(C_i)| \geq 2 \), \( p + 1 \leq j \leq q \). For every \( j \in \{p + 1, p + 2, \ldots, q\} \), let the list \( H_j \) be \( (\delta(G[C_j]), \delta(G[C_j]), \ldots, \delta(G[C_j])) \). Then there is an algorithm for computing
\[
\max_{\mathcal{S} \subseteq \mathcal{S}'} \max_{S' \cup \mathcal{S} = \mathcal{S}'} \left\{ \sum_{j=p+1}^{q} \delta_j(G[S_j]) + p : \sum_{j=p+1}^{q} r_j = k - |S'| \right\}.
\]
for every \( k \geq |S'| \) from the list \( (H_{p+1}, H_{p+2}, \ldots, H_q) \) in time \( O(n^3) \).

**Proof.** Let \( \delta(G - S')^r(p + 1 \leq r \leq q) \) be the largest number of isolated vertices in \( G \cup \delta_{\mathcal{S}'} \) after the deletion of \( k - |S'| \) vertices in \( \delta_{\mathcal{S}'} \). Thus, \( \delta(G - S')^r \) is exactly the largest number of isolated vertices that \( G \cup \delta_{\mathcal{S}'} \) can have after deleting \( k - |S'| \) vertices from \( \delta_{\mathcal{S}'} \), which is exactly
\[
\max_{\mathcal{S} \subseteq \mathcal{S}'} \max_{S' \cup \mathcal{S} = \mathcal{S}'} \left\{ \sum_{j=p+1}^{q} \delta_j(G[S_j]) + p : \sum_{j=p+1}^{q} r_j = k - |S'| \right\}.
\]

Let the list \( L^{(0)} \) be
\[
\left( \delta(G - S'), \delta(G - S'), \ldots, \delta(G - S') \right),
\]
\( p + 1 \leq r \leq q \). Then \( L^{(p+1)} = H_{p+1} \).

Furthermore, the algorithm iteratively computes for \( r = p + 2, p + 3, \ldots, q \) the list \( L^{(r)} \) from \( L^{(r-1)} \) and \( H_r \) by using
\[
\delta(G - S')^r = \max \left\{ \delta(G - S')^{r-1} + \delta(G - S') : a + b = k \right\}.
\]
The computation of an entry \( \delta(G - S')^r \) can be accomplished within time \( O(n) \). Hence, we can calculate all \( O(n^2) \) entries within time \( O(n^3) \) in the light of this algorithm.

This completes the proof.

**3**  **COMPUTING \( \delta(G) \) FOR INTERVAL GRAPHS**

We start by recalling the definition of an interval graph and providing a small example of an interval graph and its representation.

**Definition** [14] If every vertex \( y \in V(G) \) can be associated with a nonempty interval \( I_y \) on the real line, so that \((x, y) \in E(G)\) if and only if \( I_x \cap I_y \neq \emptyset \), then \( G \) is called an interval graph. In this case, the associated intervals are said to be an interval representation of \( G \).

As a matter of convenience, we use \( \Gamma \) to denote the collection of all interval graphs.

In order to intuitively express the concept of interval graph, Figure 1 illustrates an interval graph \( G \) and its representation.

Interval graphs appear quite naturally in applications, for example, in scheduling problems, as one can imagine from the above example. Many notorious NP-complete problems can be solved in polynomial time when restricted to interval graphs, that is, the vertex coloring problem. Interval graphs belong to the well-studied larger class of perfect graphs, and as such have a special structure and many nice structural properties. More information on various applications of interval graphs can be found in References 15-18.

Interval graphs have been researched extensively within structural graph theory and algorithmic graph theory. For more information and results we refer the interested reader to References 19-21. In this section, we will present a polynomial algorithm for determining \( \delta(G) \) in case \( G \in \Gamma \). We list some useful lemmas on elementary properties of interval graphs first.

**Lemma 2.** [14] Let \( G' \) be an induced subgraph of \( G \in \Gamma \). Then \( G' \in \Gamma \).
**Lemma 3.** [22] Any \((n, m)\)-graph \(G \in \Gamma\) can be recognized within \(O(m + n)\) time.

**Lemma 4.** [14] An \(n\)-vertex triangulated graph \(G\) possesses not more than \(n\) maximal cliques, where equality is achieved if and only if \(G\) is a null graph (an edgeless graph).

Here, the class of triangulated graphs (nowadays usually called chordal graphs) is a superclass of the class of interval graphs.

**Lemma 5.** [23] A graph \(G\) belongs to \(\Gamma\) if and only if the maximal cliques of \(G\) are in linear sequencing (in a clique path).

Here, a clique path of an interval graph is a sequence of all maximal cliques of \(G\), such that each edge of \(G\) is present in some clique and each vertex of \(G\) appears in consecutive cliques only. In fact, the above \(O(n + m)\) time recognition algorithm for interval graphs produces such a clique path whenever the input graph is an interval graph.

The above linear sequencing in Lemma 3 is also called a consecutive clique arrangement (CCA for short). We can determine a CCA for a graph \(G \in \Gamma\) in linear time by using the PQ-tree algorithm given by Booth and Lueker.

**Lemma 6.** [19] Let \(L_1, L_2, \ldots, L_t \leq n\) be a CCA of \(G \in \Gamma\). Then \(MC(G)\) is composed of sets \(C_i = L_i \cap L_{i+1}, s \in \{1, 2, \ldots, t - 1\}\).

**Definition 19** Let \(L_1, L_2, \ldots, L_t \leq n\) be a CCA of \(G \in \Gamma\). Suppose \(L_0 = L_{t+1} = \emptyset\). Denote \(Y(l, r) = \bigcup_{l \leq s \leq r} L_s\) for all \(l, r\) such that \(1 \leq l \leq r \leq t\). If \(G[Y(l, r)]\) is connected for \(Y(l, r) \neq \emptyset\), the set \(Y(l, r), 1 \leq l \leq r \leq t\), is known as a piece of \(G\). Moreover, \(V = Y(1, t)\) is a piece.

According to the concept of piece in \(G\), any \(G \in \Gamma\) contains two kinds of pieces. If a piece induces a complete graph, we call it complete. Otherwise, we name it incomplete. If piece \(G[Y(l, r)]\) is complete, \(1 \leq r \leq t\), we have

\[
l_I(G[Y(l, r)]) = \begin{cases} 0, & \text{if } k \in \{0, 1, 2, \ldots, |G[Y(l, r)]| - 2\} \\ 1, & \text{if } k = |G[Y(l, r)]| - 1 \end{cases}
\]

If piece \(G[Y(l, r)]\) \((1 \leq l \leq r \leq t)\) is incomplete, it possesses minimal cutsets, and for any \(k \in \{\chi(G[Y(l, r)]), \ldots, |G[Y(l, r)]| - 2\}\), the following equality holds

\[
l_I(G[Y(l, r)]) = \max_{j=1}^{q} l_j(G[C_j]) + p.
\]

where the maximum is taken over all \(C_j \cap Y(l, r) \in MC(G[Y(l, r)]), s \in \{l + 1, l + 2, \ldots, r - 1\}\), satisfying the condition that \(|C_j \cap Y(l, r)| \leq k\), and over all nonnegative integer vectors \((r_1, r_2, \ldots, r_q)\) requiring that \(\sum_{j=1}^{q} r_j = k - |C_j \cap Y(l, r)|. C_1, C_2, \ldots, C_p\) are the isolated vertices in \(G[Y(l, r)] - C_j\), and \(C_{p+1}, C_{p+2}, \ldots, C_q\) are the other components in \(G[Y(l, r)] - C_j\) such that \(|V(C_j)| \geq 2(p + 1) \leq q\).

If \(G \in \Gamma\) is complete, then \(ir(G) = \infty\). If not, then based on Theorem 3, we can calculate \(ir(G)\) of a noncomplete \(G \in \Gamma\) using Algorithm 1.

In the following theorem, we evaluate the algorithm with regard to two aspects. First of all, we prove that the algorithm is correct. Second, it is testified that the algorithm can be implemented in polynomial time.

**Theorem 5.** For any \(n\)-vertex graph \(G \in \Gamma\), \(ir(G)\) will be outputted by Algorithm 1 within time complexity \(O(n^4)\).

**Proof.** By Theorem 3 and Lemma 7, one can easily convince oneself that the algorithm is correct. Evidently, the steps in lines 2–3 can be carried out in time \(O(1)\). The steps in lines 5–11 and 19 can be implemented in time \(O(n^4)\) in an explicit way.
Algorithm 1. Algorithm Isolated Toughness

Input: An n-vertex $G \in \Gamma$ with CCA $L_1, L_2, \ldots, L_t$

Output: Isolated toughness $\tau(G) = \min\{\sum_{i=1}^{k} i \cdot \ell_i(G[Y(l, r)]) > 1\}$.

1 begin
2 $L_0 \leftarrow \emptyset$;
3 $L_{t+1} \leftarrow \emptyset$;
4 for $w \leftarrow 0$ to $t + 1$ do
5     calculate $l(y) = \min\{w : y \in L_w\}$ for every $y \in V$.
6     For all $l, r, 1 \leq l \leq r \leq t$, determine $\Upsilon(l, r)$;
7     for $l, r \in \Upsilon(l, r)$ do
8         if $\Upsilon(l, r) \neq 0$ then
9             $\Upsilon(l, r)$ is labeled "complete";
10            $\Upsilon(l, r)$ is connected then
11                label $(l, r)$ "complete";
12            end
13     end
14     for all unlabelled $(l, r)$, inspect if $G[\Upsilon(l, r)]$ is connected;
15     if $G[\Upsilon(l, r)]$ is connected then
16         $\Upsilon(l, r)$ is labeled "complete";
17     end
18 end
19
In the steps at line 12 and lines 16–18, we can use an $O(n + m)$ algorithm to check connectedness and count components for as much as $n^2$ graphs $G[\Upsilon(l, r)]$. In case $G[\Upsilon(l, r)]$ is not a connected graph and $Q_i$ is a component in $G[\Upsilon(l, r)]$, then $Q_j = \Upsilon(l, r)$, having $l(j) = \min\{l(y) : y \in Q_i\}$ and $r(j) = \max\{l(y) : y \in Q_i\}$ which will be calculated within time $O(n)$. Hence, the steps at line 12 and lines 16–18 can be carried out within $O(n^2)$ time.

The steps in lines 13–15 can be performed for not more than $n^2$ triplets $(l, r, l \leq s \leq r$ in case $\Upsilon(l, r) \neq \emptyset$. Lemma 7 shows us how to calculate components of $G[\Upsilon(l, r) \cap C(s)]$ by making use of the labels of $(l, s)$ and $(s + 1, r)$, in the event of the label is "incomplete" or "complete"; then $(l, s)$ and $(s + 1, r)$ are stored separately. If the label is "disconnected", the relevant linked list is added.

Therefore, $(l, s)$ and $(s + 1, r)$ can be calculated within time $O(n)$. By Lemma 1, it is easy to check that $G[\Upsilon(l, r) \cap C(s)] \in MC(G[\Upsilon(l, r) \cap C(s)])$ if there exist at least two components in $(l, s)$, so that every $v \in V \Upsilon(l, r) \cap C(s)$ has a neighbor in each of these two components. From the CCA, it is enough to check two components $Q_i$ of $G[\Upsilon(l, s)]$ having the two largest values of $r$, as well as two components $Q_j$ of $G[\Upsilon(s + 1, r)]$ having the two smallest values of $l$, which can be completed in time $O(n)$. Therefore, the steps in lines 13–15 can be completed in time $O(n^2)$.

The steps on lines 20–22 demand us to calculate the right-hand side of Equation (2) for each $k \in \{k \cdot \ell_i(G[Y(l, l + d)]) \colon |Y(l, l + d)| - 2\}$, using a simple table search, we can determine the list $H_k = (i_0(Q), i_1(Q), \ldots, i_{\ell_i(Q)})$, $j \in \{1, 2, \ldots, t\}$, for each component $Q_i = (l, r)$ of $G[\Upsilon(l, r) \cap C(s)]$ in time $O(n^2)$. Hence,

$$\max_{\sigma \in \Sigma_{p_1 + p_2 + \ldots + p_t = \infty}} \left\{ \sum_{j=p_1+1}^{q} i_j(G[Q]) + p : \sum_{j=p_1+1}^{q} r_j = k \cdot |Y(l, l + d)| \cap C(s) \right\}$$

can be determined in time $O(n^2)$ for a given minimal cutset $Y(l, l + d) \cap C(s)$ and for every $k$ with $|Y(l, l + d) \cap C(s)| \leq k$ in time $O(n^2)$.

Therefore, we conclude from the above analysis that the worst time complexity of Algorithm 1 is $O(n^2)$.
4 | COMPUTING \(i_r(G)\) FOR SPLIT GRAPHS

Split graphs constitute another class of graphs for which several NP-complete and NP-hard problems become solvable in polynomial time. In this section, we will show that \(i_r(G)\) can be determined in polynomial time for split graphs. We start again with some definitions and structural observations.

**Definition** [24] If the vertex set \(V\) of a graph \(G\) can be partitioned into two parts, such that one part induces an independent set \(I\) and the other part induces a clique \(C\) in \(G\), then \(G\) is called a split graph.

We will use \(G = (C, I, E)\) to represent a split graph according to the above definition. To avoid trivial cases, and noting that we can use our approach to compute the isolated toughness of each component of a disconnected split graph, we suppose throughout this section that \(G\) is connected, \(C \neq \emptyset\), and \(I \neq \emptyset\).

If \(N(I) \neq C\), we can select a vertex \(v \in C \setminus N(I)\), and replace \(C\) by \(C \setminus \{v\}\) and replace \(I\) by \(I \cup \{v\}\), such that we obtain an alternative representation \(G = (C \setminus \{v\}, I \cup \{v\}, E)\) for our split graph \(G\). For this new representation, clearly \(N(I) \cup \{v\} = C \setminus \{v\}\). Based on this observation, we will suppose in the sequel that \(N(I) = C\) in any split graph \(G = (C, I, E)\) that we consider as an input graph.

By transforming the problem of calculating the toughness \(\tau(G)\) for a split graph to a submodular function minimization problem, Woeginger [25] was able to show that this toughness problem can be solved in polynomial time for split graphs. Zhang et al. [26] corroborated in a more recent paper that by an analogous approach the scattering number of a split graph can be determined in polynomial time. Three of the authors of the current paper proved in Reference 27 that the isolated scattering number of split graphs can be computed in polynomial time as well by adapting the technique that was used in Reference 26. We need a few more definitions and notations in order to explain how we can use a similar approach for determining the isolated toughness.

For any finite set \(\Lambda\), a real function \(g : 2^\Lambda \rightarrow \mathbb{R}\) on all subsets of \(\Lambda\) is called submodular [28] in case

\[
g(S \cap T) + g(S \cup T) \leq g(S) + g(T)
\]

holds for all \(S, T \subseteq \Lambda\).

Submodular functions appear in many different fields of mathematics and their application areas, for instance, as rank functions of (poly)matroids, but also in matroid intersection, submodular flow problems, and as objective functions in algorithmic game theory and machine learning. They are particularly useful in minimization problems because of the seminal result due to Schrijver [29] who developed a strongly polynomial time algorithm for solving minimization problems involving submodular functions. For our purpose we need a slight adaptation to a so-called crossing family of subsets.

Let \(\Re\) be a family of subsets of \(\Lambda\). Then \(\Re\) is said to be a crossing family [28] if the following implication holds:

\[
S, T \in \Re, S \cap T \neq \emptyset, S \cup T \neq \Lambda \Rightarrow S \cap T \subseteq \Re, S \cup T \subseteq \Re.
\]

The problem of minimizing a submodular function on a crossing family can be transformed into \(|\Lambda|(|\Lambda| - 1)\) problems of finding minimum values of submodular functions on \(2^\Lambda\) [28]. It is obvious that all proper subsets \(\emptyset \neq \emptyset \neq \emptyset \subseteq \Lambda\) constitute a crossing family. Thus, if \(g\) is a submodular function having domain \(2^\Lambda \setminus \{\emptyset, \emptyset\}\), then \(\min\{g(V) : V \neq \emptyset, V \subseteq \Lambda\}\) can be calculated in polynomial time too. We use this to prove our other main result.

**Theorem 6.** The isolated toughness \(i_r(G)\) of a noncomplete connected split graph can be calculated in polynomial time.

**Proof.** Let \(G = (C, I, E)\) be a connected \(n\)-vertex split graph which is not complete. Let \(\theta\) be some given rational number. We are going to show that the problem of determining \(i_r(G)\) boils down to checking whether there is a vertex set \(\Omega \subseteq I\) so that \(|N(\Omega)| \leq \theta \cdot |\Omega|\). Once this has been shown we can consider the function \(g(V) = |N(V)| - \theta \cdot |V|\) with domain \(2^\Lambda \setminus \{\emptyset, \emptyset\}\). It is easy to verify that \(g(V)\) is a submodular function whose domain is the crossing family \(2^\Lambda \setminus \{I, \emptyset\}\). From this and the above arguments, we conclude that calculating \(\min\{g(V) : V \neq \emptyset, V \subseteq I\}\) can be done in polynomial time too.

We complete the proof by showing that our problem indeed boils down to checking whether there is a vertex set \(\Omega \subseteq I\) with \(|N(\Omega)| < \theta \cdot |\Omega|\). For determining the isolated toughness, one has to verify whether or not \(G\) has a cutset \(H^+ \subseteq C(G)\) with cardinality \(|H^+| < \theta \cdot |G - H^+|\). If there is such a cutset \(H^+\), then w.l.o.g. we may assume that \(H^+ \subseteq \emptyset\); if this is not the case, we can simply replace \(H^+\) by \(H^+ \cap \emptyset\), noting that this would not decrease \(i_r(G - H^+)\), and simultaneously this does not increase \(|H^+|\). Now we distinguish two cases:

**Case 1.** \(H^+ = \emptyset\). Then we have \(i_r(G - H^+) = |I|\), and it is trivial to check whether \(|H^+| < \theta \cdot |G - H^+|\).

**Case 2.** \(H^+ \subseteq I\). Then \((G - H^+) = \emptyset\), where \(\emptyset = \emptyset \setminus i_r(G) \subseteq \emptyset\). We next prove that \(|H^+| < \theta \cdot |G - H^+|\) is equivalent to \(|N(\Omega)| < \theta \cdot |\Omega|\), as follows. First, let \(|H^+| < \theta \cdot |G - H^+|\). Then it is clear that \(|N(\Omega)| \leq |H^+|\) and \(i_r(G - H^+) = |\Omega|\). Hence,

\[
\frac{|N(\Omega)|}{|\Omega|} \leq \frac{|H^+|}{i_r(G - H^+)} < \theta,
\]

that is, \(|N(\Omega)| < \theta \cdot |\Omega|\).
For the reverse, let \( \Omega \subseteq I \) such that \( |N(\Omega)| < \theta \cdot |\Omega| \), and let \( H^* = N(\Omega) \). Then \( i(G - H^*) \geq |\Omega| \). Thus,

\[
\frac{|H^*|}{|G - H^*|} \leq \frac{|N(\Omega)|}{|\Omega|} < \theta.
\]

This confirms that the problem boils down to the question whether there exists a vertex set \( \Omega \subseteq I \) with the property that \( |N(\Omega)| < \theta \cdot |\Omega| \).

To complete the proof, we point out that \( ir(G) \) of a graph \( G = (V, I, E) \) is a positive rational number whose denominator and numerator are both bounded from above by \( n \). Hence, one can run through all possible choices for \( \theta \) and inspect whether \( ir(G) < \theta \) in polynomial time.

This completes the proof.

5 | CONCLUSION

The research that led to this article deals with the computational complexity of determining the isolated toughness of interval graphs and split graphs. The work is strongly motivated by the role the isolated toughness plays in many key existence results regarding factors and fractional factors in graphs. These factors have many applications in various areas, such as network design, scheduling and combinatorial topology. In this article, we have presented a polynomial algorithm to compute the isolated toughness for interval graphs. Moreover, we have shown that the isolated toughness of split graphs can also be calculated in polynomial time, by transforming it to a minimization problem for submodular functions.

ACKNOWLEDGMENTS

This work was supported by NSFC (No. 11871280), Natural Science Foundation of Zhejiang Province (China) (No. LY17A010017), and Qing Lan Project.

CONFLICT OF INTEREST

The authors declared that they have no conflicts of interest to this work.

ORCID

Xiaoyan Zhang https://orcid.org/0000-0002-2224-1484

REFERENCES


How to cite this article: Li F, Ye Q, Broersma H, Zhang X. Polynomial algorithms for computing the isolated toughness of interval and split graphs. *Concurrency Computat Pract Exper*. 2021;e6345. [https://doi.org/10.1002/cpe.6345](https://doi.org/10.1002/cpe.6345)