On the Independent Set and Common Subgraph Problems in Random Graphs

Yinglei Song
School of Computer Science and Engineering, Jiangsu University of Science and Technology
Zhenjiang, Jiangsu 212003, China
yingleisong@gmail.edu

Abstract
In this paper, we develop efficient exact and approximate algorithms for computing a maximum independent set in random graphs. In a random graph \( G \), each pair of vertices are joined by an edge with a probability \( p \), where \( p \) is a constant between 0 and 1. We show that, a maximum independent set in a random graph that contains \( n \) vertices can be computed in expected computation time \( 2^{O(\log^2 n)} \). Using techniques based on enumeration, we develop an algorithm that can find a largest common subgraph in two random graphs in \( n \) and \( m \) vertices (\( m \leq n \)) in expected computation time \( 2^{O(n^{1/2} \log^2 n)} \). In addition, we show that, with high probability, the parameterized independent set problem is fixed parameter tractable in random graphs and the maximum independent set in a random graph in \( n \) vertices can be approximated within a ratio of \( \frac{2n}{2n - \log n} \) in expected polynomial time.

1 Introduction
In computer science, many optimization problems can be reduced to the optimization of objectives that are formulated and described in a graph. The development of efficient exact or approximate algorithms for graph optimization problems thus constitute an important part of the research in combinatorial optimization. However, a large number of graph optimization problems have been shown to be NP-hard \[18\], which suggests that it is unlikely to develop algorithms that can solve these problems in polynomial time. A well known example is the Maximum Independent Set problem. Given a graph \( G = (V, E) \), a vertex set \( I \subseteq V \) is an independent set if there is no edge between any pair of two vertices in \( I \). The goal of the Maximum Independent Set problem is to find an independent set of the largest size in a given graph \( G \). The problem can be trivially solved in time \( 2^{O(n)} \) by enumerating and checking all possible vertex subsets in the graph. Although intensive research has been performed to improve the computation time needed to find an optimal solution \[2, 6, 14, 17, 30, 32, 33, 34, 37, 38\], an algorithm that needs subexponential time is not yet available for this problem. Recently, it is proposed that this problem is unlikely to be solved in subexponential time \[7, 8\].

Due to the difficulty of developing efficient algorithms that can find optimal solutions for these problems, a large number of algorithms have been developed to generate approximate solutions that are close to optimal ones in polynomial time \[24\]. Solutions provided by these algorithms are often guaranteed to be within a ratio of the optimal solution and thus can be useful in practice. For example, the Minimum Vertex Cover problem can be approximated by a simple polynomial time algorithm within a ratio of 2.0. However, it has been shown that it is NP-hard to approximate this problem within a ratio of 1.362 \[9\]. A well known inapproximability result regarding the Maximum Independent Set problem is that it is NP-hard to approximate the maximum independent set in a graph within a ratio of \( n^{1-\epsilon} \), where \( 0 < \epsilon < 1 \) is a constant and \( n \) is the number of vertices in the graph \[21\]. This result suggests that an approximate solution with a guaranteed constant approximate ratio cannot be obtained in polynomial time for the Maximum Independent Set problem.
problem unless NP=P. So far, the best known approximation ratio that has been achieved for this problem in general graphs is $O\left(\frac{n \log^2 \log_2 n}{\log_2 n}\right)$ \cite{15}.

For those problems that cannot be even approximated within a good approximation ratio in polynomial time, such as the MAXIMUM INDEPENDENT SET problem, heuristics that can efficiently generate approximate solutions are often employed in practice to solve them \cite{3 20 21}. However, solutions generated by heuristics are not guaranteed to be close to the optimal ones and their applications are thus restricted to scenarios where the accuracy of solutions is not a crucial issue.

Parameterized computation provides another potentially practical solution for some problems that are computationally intractable. In particular, one or a few parameters in some intractable problems can be identified and parameterized computation studies whether efficient algorithms exist for these problems while all parameters are small. A parameterized problem may contain a few parameters $k_1, k_2, \ldots, k_l$ and the problem is fixed parameter tractable if it can be solved in time $O(f(k_1, k_2, \ldots, k_l)n^c)$, where $f$ is a function of $k_1, k_2, \ldots, k_l$, $n$ is the size of the problem and $c$ is a constant independent of all parameters. For example, the MAXIMUM INDEPENDENT SET problem is to determine whether a graph $G = (V, E)$ contains a vertex cover of size at most $k$ or not. The problem is NP-complete. However, a simple parameterized algorithm can solve the problem in time $O(2^k|V|)$ \cite{11}. In practice, this algorithm can be used to efficiently solve the VERTEX COVER problem when the parameter $k$ is fixed and small. On the other hand, some problems do not have known efficient parameterized solutions and are therefore parameterized intractable. Similar to the conventional complexity theory, a hierarchy of complexity classes has been constructed to describe the parameterized complexity of these problems \cite{11}. For example, the INDEPENDENT SET problem is to decide whether a graph contains an independent set of size $k$ or not and has been shown to be W[1]-complete \cite{12}. It cannot be solved with an efficient parameterized algorithm unless all problems in W[1] are fixed parameter tractable. A thorough investigation on these parameterized complexity classes are provided in \cite{10}.

In this paper, we develop exact and approximate algorithms for the MAXIMUM INDEPENDENT SET problem where the underlying graph is a random graph generated based on the Erdős–Rényi model \cite{13}. Such a random graph is generated by treating each pair of vertices independently and adding an edge to join them with a probability of $p$ ($0 < p < 1$), where $p$ is a constant. Recent research in molecular biology has shown that the protein side chain interaction network conforms remarkably well to random graphs generated by the Erdős–Rényi model \cite{5}. Therefore, efficient algorithms for some NP-hard problems in random graphs, if exist, may significantly improve the computational efficiency for some important optimization problems related to protein structure prediction.

In \cite{19} \cite{25}, it has been shown that with high probability, the maximum independent set in a random graph is of size $O(\log_2 n)$. However, this result does not directly lead to an algorithm that can compute the maximum independent set in a random graph in expected subexponential time. In \cite{19}, a polynomial time algorithm that can compute a maximum independent set in a sparse random graph with high probability is developed. However, the algorithm is based on a large independent set that is embedded in the graph and thus cannot be used for all graphs. We show that the maximum independent set in a random graph can be computed in expected computation time $2^{O(\log^2 n)}$, where $n$ is the number of vertices in the graph. This result significantly improves the best known time complexity $O(2^\sqrt{\log n})$ for finding a maximum independent set in general graphs \cite{51}.

Using techniques based on enumeration, we develop an algorithm that can compute a largest common subgraph of two random graphs of $n$ and $m$ vertices ($n \geq m$) in expected computation time $2^{O(n \log^2 n)}$. This result significantly improves on the best known time complexity $2^{O(m \log n)}$ for this problem when $m = O(n)$. In addition, we show that, with high probability, the parameterized independent set problem is fixed parameter tractable in random graphs. For approximate algorithms, we develop an algorithm that can achieve an approximation ratio of $\frac{1}{2^{\sqrt{\log n}}} \sqrt{2n}$ in expected polynomial time, which is a significant improvement compared with the best known approximate ratio that can be achieved in general graphs \cite{11} 35.
2 Maximum Independent Set in Random Graphs

A random graph $G(V, p)$, where $0 < p < 1$, is a graph obtained by independently adding edges between each pair of vertices in $V$ with a probability $p$. Given a vertex $v \in V$, the degree of $v$ in $G$ is the number of vertices that are connected to $v$ by an edge in $G$. We use $\deg_G(v)$ to denote the degree of vertex $v$ in graph $G$ and $N_G(v)$ to denote the set of vertices that are connected to $v$ by an edge in $G$. A vertex subset $I \subseteq V$ is an independent set in $G$ if there is no edge between any pair of vertices in $I$. The goal of the Maximum Independent Set problem is to find an independent set of the largest size in a given graph.

In [19, 25], it is shown that, with high probability, the size of a maximum independent set in a random graph $G(V, p)$ is $2 \log_2 \frac{n}{\log_2 \frac{1}{1-p}}$, where $n$ is the number of vertices in $G$. A straightforward algorithm by exhaustively enumerating all vertex subsets of size $2 \log_2 \frac{n}{\log_2 \frac{1}{1-p}}$ can thus compute a maximum independent set in most random graphs in time $n \log_2 n$. However, to compute a maximum independent set in all random graphs, the algorithm must be able to cope with the cases where the graph contains an independent set of size larger than $O(\log_2 n)$. The algorithm needs time $2^{O(n)}$ to compute a maximum independent set in these cases. The best known upper bound of the probability for a random graph to have a maximum independent set larger than $O(\log_2 n)$ is $\frac{1}{n^{O(1)}}$ [19, 25], the expected time complexity of this enumeration based algorithm is thus $2^{O(n)}$.

We show that the maximum independent set in a random graph $G = (V, p)$ can be computed in expected subexponential time.

**Lemma 2.1** Given a random graph $G = (V, p)$ where $n = |V|$ and a sufficiently small constant $\epsilon$ such that $\epsilon < p$, there exists a vertex $v \in V$ such that $\deg_G(v) \geq (p - \epsilon)n$ with probability at least $1 - 2^{-\mu n^2}$, where $\mu$ is a positive constant that only depends on $\epsilon$ and $p$.

**Proof.** If such a vertex does not exist, the number of edges $n(E)$ in $G$ is at most $\frac{(p-\epsilon)n^2}{2}$ since the degree of each vertex is at most $(p - \epsilon)n$. However, from the construction of graph $G$, the expected number of edges in $G$ can be obtained as follows

$$E(n(E)) = \frac{pn(n-1)}{2}$$

From Chernoff bound, we can bound the probability for $n(E) < \frac{(p-\epsilon)n^2}{2}$ by

$$Pr(n(E) < \frac{(p-\epsilon)n^2}{2}) < \exp\left(-\frac{pn(n-1)\delta^2}{4}\right)$$

where $\delta = \frac{\epsilon(n-1)}{p(n-1)}$. For sufficiently large $n$, we have

$$\delta > \frac{\epsilon}{2p}$$

$$n - 1 > \frac{n}{2}$$

We can thus immediately obtain

$$Pr(n(E) < \frac{(p-\epsilon)n^2}{2}) < \exp\left(-\frac{\epsilon^2 n^2}{32p}\right) = 2^{-\frac{\epsilon^2 n^2}{32p}}.$$ 

We then let $\mu = \frac{\epsilon^2}{32p}$ and we conclude that with probability at least $1 - 2^{-\mu n^2}$, there exists vertex $v \in V$ such that $\deg_G(v) \geq (p - \epsilon)n$. 


The proof of Lemma 2.1 relies on the fact that $p$ is a constant independent of $n$, the Lemma does not hold if the value of $p$ depends on $n$. A random graph $G = (V, p)$ in $n$ vertices is good if it contains at least one vertex whose degree is at least $(p-\epsilon)n$. Given a random graph, the algorithm starts by finding a vertex $v$ such that $deg_G(v)$ is at least $(p-\epsilon)n$. If such a vertex does not exist, the algorithm enumerates all subsets of $V$ and returns an independent set of the largest size. If $v$ exists, the algorithm branches on two possible cases on whether $v$ is contained in $I$ or not. In particular, if $v \in I$, $v$ and vertices in $N(v)$ are deleted from $G$ and the resulting graph is $G_1$; if $v \notin I$, $v$ is deleted from $G$ and the resulting graph is $G_2$. The algorithm is then recursively applied on both $G_1$ and $G_2$ to compute a maximum independent set in each of them. We use $I_1$ and $I_2$ to denote the maximum independent sets in $G_1$ and $G_2$ found by the algorithm respectively. $I_2$ is returned as a maximum independent set in $G$ if $|I_2| \geq |I_1| + 1$ and $I_1 \cup \{v\}$ is returned otherwise. We show that this algorithm terminates in expected time $2^{O(\log^2 \Delta n)}$.

Theorem 2.1 A maximum independent set in a random graph $G = (V, p)$ with $n$ vertices can be computed in expected computation time $2^{O(\log^2 \Delta n)}$.

Proof. We show that the algorithm described above terminates in expected time $2^{O(\log^2 \Delta n)}$. In particular, the algorithm is recursive and for each step of recursion, we have the following recursion relation for the computation time if the underlying graph is good and contains $m$ vertices

$$T(m) \leq T((1-p+\epsilon)m) + T(m-1) + O(m^2) \tag{8}$$

where $T(m)$ is the computation time needed by the algorithm in a graph on $m$ vertices. The term $O(m^2)$ is the computation time needed to find a vertex whose degree is at least $(p-\epsilon)m$, since the time needed to compute the degree of a vertex is $O(m)$ and the algorithm may need to check $m$ vertices to find such a vertex. If the underlying graph is not good, the algorithm exhaustively enumerates all subsets in the graph and finds an independent set of the largest size. The computation time is $2^{O(m)}$.

We are now ready to establish the expected computation time for the algorithm. In particular, we use $ET(m)$ to denote the expected computation time of the algorithm on a graph that contains $m$ vertices. From Lemma 2.1 an underlying graph $G'$ in $m$ vertices is good with a probability of at least $1-2^{-\mu m^2}$. We thus can immediately obtain the following recursion for $ET(m)$.

$$ET(m) \leq ET((1-p+\epsilon)m) + ET(m-1) + O(m^2) + 2^{O(m)-\mu m^2} \tag{9}$$

$$ET((1-p+\epsilon)m) + ET(m-1) + O(m^2) \tag{10}$$

where the second inequality is due to the fact that $2^{O(m)-\mu m^2}$ is bounded by a constant for all positive integers $m$.

We then show that $ET(m) \leq 2^{c \log^2 \Delta m}$, where $c$ is a positive constant. We show this by induction. First, for a sufficiently large positive integer $m_0$ whose value will be specified later, we let $c_0 = \max_{1 \leq t \leq m_0} \left(\log_{2 \Delta} ET(t)\right)$ and choose $c = \max \{c_0, \frac{2}{1-p+\epsilon}\}$. It is not difficult to see that $ET(l) \leq 2^{c \log^2 \Delta l}$ if $1 \leq l \leq m_0$. We then assume this holds for all positive integers less than $m$. From the above recursion relation on $ET(m)$, we can obtain

$$ET(m) \leq 2^{c \log^2 \Delta ((1-p+\epsilon)m)} + 2^{c \log^2 \Delta (m-1)} + Bm^2 \tag{11}$$

$$\leq sm^{-2^{c \log^2 \Delta m}} + 2^{c \log^2 \Delta m} + (2^{c \log^2 \Delta (m-1)} - 2^{c \log^2 \Delta m}) + Bm^2 \tag{12}$$

$$\leq sm^{-2^{c \log^2 \Delta m}} + 2^{c \log^2 \Delta m} - \log_2 m \log_2 m + Bm^2 \tag{13}$$

$$\leq 2^{c \log^2 \Delta m} \tag{14}$$

where $B$ is a positive constant independent of $c, p, \epsilon$ and $s, q, l$ are some positive constants that depend on $c, p, \epsilon$ only. The first inequality is obtained from the assumption for induction. The second one is due to the fact that $\log_2 \left( (1-p+\epsilon)m \right) = \log_2 \left( (1-p+\epsilon) \log_2 m \right)$ and we can let $l = 2c \log_2 \frac{1}{1-p+\epsilon}$, $s = 2^{c \log^2 (1-p+\epsilon)}$. 

To establish the third inequality, we have
\[
\log_2 (m - 1) - \log_2 m = \left( \log_2 m + \frac{1}{m} \right)^2 - \log_2 m
\]
\[
\leq \left( \log_2 m - \frac{1}{6m} \right)^2 - \log_2 m
\]
\[
\leq -\frac{\log_2 m}{6m}
\]
when \( m \geq 16 \), we can obtain
\[
2^{c \log_2 (m - 1) - m} = 2^{c \log_2 m (2^{c \log_2 m} - 2) - 1)
\]
\[
\leq m \left( -\log_2 m \right) \leq 24m
\]
the third inequality thus follows.

From the fact that \( c \geq \frac{2}{\log_2 e} \), we have \( l \geq 4 \). We let
\[
c' = \log_2 \left( \frac{2}{1-p+e} \right)
\]
\[
s' = 2^{c \log_2 (1-p+e)}
\]
\[
l' = 2^{c \log_2 \left( \frac{1}{1-p+e} \right)}
\]
we now consider the function \( F(m) = (s'm^{-l'} - \log_2 m)^2 + Bm^2 \). Since \( s', l', c \), and \( B \) are independent of \( m \) and \( l' \geq 4 \), there exists a positive integer \( m_1(p, e) \) such that \( F(m) \leq 0 \) when \( m \geq m_1(p, e) \). \( m_0 \) can be determined as follows
\[
m_0 = \max \{ m_1(p, e), \frac{1}{\sqrt{1-p+e}}, 16 \}
\]

It is not difficult to see that when \( c \geq c' \) and \( m \geq m_0 \), we have \( s'm^{-l'} - \log_2 \frac{m}{24m} \leq 0 \). In addition, we can further verify that
\[
s'm^{-l'} = 2^{c \log_2 (1-p+e) \log_2 (m^2(1-p+e))}
\]
\[
s'm^{-l'} \leq 2^{c \log_2 (1-p+e) \log_2 (m^2(1-p+e))}
\]
the following thus holds
\[
(s'm^{-l'} - \log_2 \frac{m}{24m})^2 \leq (s'm^{-l'} - \log_2 \frac{m}{24m})^2 + Bm^2
\]
the fourth inequality thus follows. From the principle of induction, the theorem has been proved.

\[
\square
\]
3 Parameterized Algorithm for Independent Set Problem

The parameterized independent set problem is to decide whether a given graph $G = (V, E)$ contains an independent set of size $k$ or not. The problem is known to be $W[1]$-hard \cite{10,11,12} and cannot be solved in time $n^{o(k)}$ in general graphs unless $W[2]=\text{FPT}$ \cite{7,8}. We show that if the underlying graph $G$ is a random graph, the problem can be solved in expected time $2^{O(k^2)} + O(n^3)$, where $n$ is the number of vertices in the graph. We need the following lemma to analyze the time complexity of the algorithm.

Lemma 3.1 Given a random graph $G = (V, p)$ where $n = |V|$ and a sufficiently small constant $\epsilon$ such that $p + \epsilon < 1$, there exists vertex $u \in V$ such that $\text{deg}_G(u) \leq (p + \epsilon)n$ with a probability of at least $1 - 2^{-\mu n^2}$, where $\mu$ is a positive constant that only depends on $\epsilon$ and $p$.

**Proof.** The proof is similar to the proof of Lemma \ref{lem1}. If such a vertex does not exist, the degree of every vertex in $G$ is at least $(p + \epsilon)n$. The graph thus contains at least $\frac{(p + \epsilon)n^2}{2}$ edges. The expected number of edges in $G$ is $\frac{pn(n-1)}{2}$. We use $n(E)$ to denote the number of the edges in $G$. From Chernoff bound, we can bound the probability for $G$ to contain at least $\frac{(p + \epsilon)n^2}{2}$ edges.

\[
Pr(n(E) \geq \frac{(p + \epsilon)n^2}{2}) \geq \frac{(p + \epsilon)n^2}{2} \quad (35)
\]

\[
< \exp\left(-\frac{\epsilon^2 n^2}{64p}\right) \quad (36)
\]

\[
= 2^{-\frac{\epsilon^2 n^2}{64p n^2}} \quad (37)
\]

the lemma immediately follows by letting $\mu = \frac{\epsilon^2}{64np n^2}$.

The proof of Lemma \ref{lem3.1} relies on the fact that $p$ is a constant independent of $n$, the Lemma does not hold if the value of $p$ depends on $n$.

**Theorem 3.1** Given a random graph $G = (V, p)$, there exists an algorithm that can decide whether $G$ contains an independent set of size $k$ in expected time $2^{O(k^2)} + O(n^3)$.

**Proof.** We start the proof by comparing the values of $k$ and $L(n) = \frac{1}{2} \log \frac{1}{1 - p}$, if $k > L(n)$, we can enumerate all possible vertex subsets of size $k$ in $G$ and check whether one of them is an independent set of size $k$ or not. The enumeration and checking needs at most $O(k^2 n^k)$ time. However, since $k > L(n)$, we can obtain $n < \left(\frac{1}{1 - p - \epsilon}\right)^{3k}$, the computation time needed to determine whether $G$ contains an independent set of size $k$ or not is thus at most $O(k^2 \left(\frac{1}{1 - p - \epsilon}\right)^{3k^2}) = 2^{O(k^2)}$ in this case.

We then consider the case where $k \leq L(n)$. We use the following procedure to generate an independent set $I$. We start with the vertex $u$ with the minimum degree in $G$, we include $u$ in $I$ and remove $u$ and all its neighbors in $G$ from $G$. We denote the resulting graph by $G_1$. The procedure can be repeatedly executed until there are at most $n^{\frac{3}{2}}$ vertices left in the graph. We use $G_0 = G, G_1, G_2, G_3, \cdots, G_l$ to denote the intermediate graphs generated during this iterative procedure. It is not difficult to see that vertices in $I$ form an independent set in $G$.

We show that the above procedure can generate an independent set $I$ of size at least $L(n)$ with high probability. We use $G_1, G_2, G_3, \cdots, G_l$ to denote the resulting graph in each iterative step and $n(G_i)$ to denote the number of vertices in graph $G_i$. From Lemma \ref{lem3.1}, the following holds with a probability of at least $1 - 2^{-\mu n^2}$ for each $i$ between $0$ and $l$.

\[
n(G_{i+1}) \geq (1 - p - \epsilon)n(G_i) \quad (38)
\]

Since $n(G_i) > n^{\frac{3}{2}}$, the probability for this inequality to hold for all $i$’s between $0$ and $l$ is at least
1 - n^{2-\mu n^\frac{2}{3}}. If this inequality holds for all i’s between 0 and l. We can immediately obtain

\[
    l \geq \log_{1 - \frac{1}{\beta - \epsilon}} \left( \frac{n}{n^\beta} \right) = \frac{1}{3} \log_{1 - \frac{1}{\beta - \epsilon}} n = L(n)
\]

I thus contains at least L(n) vertices. With a probability of at least 1 - n^{2-\mu n^\frac{2}{3}}, the above iterative procedure generates an independent set of size L(n). Since k < L(n), the algorithm returns “yes” if I indeed contains L(n) independent vertices, otherwise, the algorithm simply enumerates all vertex subsets in G and checks whether one of them is an independent set of size at least k. Since the procedure for generating I needs O(n^3) time, the expected computation time needed for this is at most

\[
    O(n^3)(1 - n^{2-\mu n^\frac{2}{3}}) + 2^{O(n)}n^{2-\mu n^\frac{2}{3}} = O(n^3)
\]

where the equality is due to the fact that the second term is bounded by a constant when n is sufficiently large. The algorithm thus needs an expected time 2^{O(k^2)} + O(n^3), the theorem has been proved.

4 The Largest Common Subgraph Problem

Given two graphs G, H, a common subgraph of G and H is a third graph K such that both G and H contain an induced subgraph that is isomorphic to K. The largest common subgraph problem is to compute a common subgraph that contains the largest number of vertices. The problem has important applications in computational biology. For example, it is often desirable to identify common subgraphs in the protein interaction networks of two homologous organisms since proteins in these common subgraphs often together play important roles for certain biological functions [28]. Unfortunately, the problem is NP hard when both of the underlying graphs are general graphs [18]. The asymptotically best known algorithm for this problem needs time \( O^{*}\((m + 1)^n\) [1, 35] and little progress has been made to improve the asymptotical time complexity of this problem. We show that, given two random graphs G and H in n and m vertices, where n \( \geq m \), the largest common subgraph problem in G and H can be computed in expected time \( 2^{O(n^{\frac{3}{2} \log_2 n})} \).

**Lemma 4.1** The largest common subgraph problem can be solved in computation time \( O(m^{2m}n^m) \leq 2^{hm \log_2 n} \), where h is some positive constant that does not depend on n or m.

**Proof.** We can solve the largest common subgraph problem with the following simple algorithm. For each positive integer l not greater than m, we enumerate all vertex subsets that contain l vertices in G. For each such vertex subset S1, we enumerate all vertex subsets of size l in graph H and for each such vertex subset S2, we enumerate all possible one to one mappings between vertices in S1 and those in S2. We then check whether there exists a one to one mapping that can establish the isomorphism between the subgraph induced by S1 in G and the subgraph induced by S2 in H. The algorithm can find all common subgraphs and return one that is of the largest size.

The number of vertex subsets of size l in G is \( \binom{n}{l} \) and the number of vertex subsets of size l in H is \( \binom{m}{l} \). The number of one to one mappings between S1 and S2 is l! and the computation time needed to check whether the two subgraphs induced by S1 and S2 are isomorphic under a particular mapping is at most \( O(l^2) \). The total computation time needed to find and return the largest common subgraph is thus at most

\[
    \sum_{l=1}^{m} C\left( \binom{n}{l} \right) \left( \binom{m}{l} \right) l! \leq \sum_{l=1}^{m} Cn^m \left( \binom{m}{l} \right) m^2 \\
    \leq C2^m n^m m^2 \\
    \leq 2^{hm \log_2 n}
\]

The proof is complete. \( \square \)
where \( C \) and \( h \) are some positive constants independent of \( n \) and \( m \). The first inequality is due to the fact that \( \binom{n}{k} \geq n^k \) and \( l \leq m \); the second inequality is due to the fact that \( \sum_{l=1}^{m} \binom{m}{l} = 2^m - 1 \). The lemma thus has been proved.

**Lemma 4.2** Given two random graphs \( G = (V, p) \) and \( H = (U, q) \), where \( p \) and \( q \) are positive constants between 0 and 1, \( G \) contains \( n \) vertices and \( H \) contains \( m \) vertices \((n \geq m)\), the probability that \( G \) and \( H \) contain a common subgraph of size \( n^\frac{2}{3} \log_2^2 n \) is at most \( 2^{-\mu n \log_2^2 n} \), where \( \mu \) is a positive constant that only depends on \( p \) and \( q \).

**Proof.** We let \( k = n^\frac{2}{3} \log_2^2 n \) and consider two given subsets of size \( k \) in graph \( G \) and \( H \) respectively. We use \( S_1 = \{g_1, g_2, \cdots, g_k\} \) and \( S_2 = \{h_1, h_2, \cdots, h_k\} \) to denote them and \( G_1, H_1 \) to denote the subgraphs induced by them in \( G \) and \( H \) respectively. We assume that \( G_1 \) is isomorphic to \( H_1 \) under a given one to one mapping \( M \), where vertex \( g_i \) in \( S_1 \) is mapped to \( h_i \) in \( S_2 \) for \( 1 \leq i \leq k \).

We then estimate the probability for \( M \) to be such a mapping. If \( G_1 \) is isomorphic to \( H_1 \) under \( M \), for any integer pair \((i, j)\), where \( 1 \leq i < j \leq k \), either both \((g_i, g_j)\) and \((h_i, h_j)\) are edges or neither of them are edges. The probability for the former case is \( pq \) and the probability for the latter case is \((1-p)(1-q)\). Since there are in total \( \binom{k}{2} \) such pairs, the probability for \( G_1 \) and \( H_1 \) to be isomorphic under \( M \) is thus \( (pq + (1-p)(1-q) \frac{k(k-1)}{2})^\frac{k(k-1)}{2} \).

We use \( P(k) \) to denote the probability for \( G \) and \( H \) to contain a common subgraph of size \( k \). Since the number of vertex subsets of size \( k \) in \( G \) is \( \binom{n}{k} \) and the number of vertex subsets of size \( k \) in \( H \) is \( \binom{m}{k} \), we can obtain an upper bound for \( P(k) \) using the union bound.

\[
P(k) \leq \binom{n}{k} \binom{m}{k} \sum_{M} s^\frac{k(k-1)}{2} \tag{46}
\]

\[
\leq \binom{n}{k} \binom{m}{k} k! s^\frac{k(k-1)}{2} \tag{47}
\]

\[
\leq n^k m^k s^\frac{k(k-1)}{2} \tag{48}
\]

\[
\leq n^2k s^\frac{k(k-1)}{2} \tag{49}
\]

\[
\leq 2^{n^\frac{2}{3} \log_2^2 n} s^\frac{k(k-1)}{2} \tag{50}
\]

\[
= 2^{n^\frac{2}{3} \log_2^2 n} n^\frac{k(k-1)}{2} \tag{51}
\]

\[
\leq 2^{-\mu n \log_2^2 n} \tag{52}
\]

where \( s = pq + (1-p)(1-q) \) and \( \mu \) is some positive constant that depends on \( p \) and \( q \) only. The first inequality is due to the union bound. The second inequality follows from the fact that there are in total \( k! \) one to one mappings between vertices in \( S_1 \) and \( S_2 \). The third inequality is due to the fact that \( \binom{n}{k} \leq n^k \) and \( \binom{m}{k} \leq m^k \). The fifth inequality follows from the fact that \( k = n^\frac{2}{3} \log_2^2 n \) and \( k^\frac{k(k-1)}{2} > \frac{k^2}{8} \) when \( n \) is sufficiently large. The last inequality is due to the fact that \( s < 1 \) and \( 2n^\frac{2}{3} \log_2^2 n \leq \frac{n \log_4 n \log_2^2 n}{8} \) for sufficiently large \( n \).

The proof of the Lemma 4.2 relies on the fact that \( p \) and \( q \) are both constants independent of \( n \) and \( m \), the Lemma does not hold if the values of \( p \) and \( q \) depend on \( n \) or \( m \).

**Theorem 4.1** Given two random graphs \( G = (V, p) \) and \( H = (U, q) \), where \( p \) and \( q \) are positive numbers between 0 and 1, \( G \) contains \( n \) vertices and \( H \) contains \( m \) vertices \((m \leq n)\), a largest common graph of \( G \) and \( H \) can be computed in expected time \( 2^{O(n^\frac{2}{3} \log_2^2 n)} \).

**Proof.** We only need to show that such an algorithm exists when \( m > n^\frac{2}{3} \log_2^2 n \). Since if \( m \leq n^\frac{2}{3} \log_2^2 n \), the algorithm in the proof of Lemma 4.1 can be directly used to find a largest common subgraph of \( G \) and \( H \) in time \( 2^{O(n^\frac{2}{3} \log_2^2 n)} \).
Let \( k = n^{\frac{1}{2}} \log^{\frac{1}{2}} n \), since \( m > k \), we can use the following algorithm to compute a largest common subgraph in \( G \) and \( H \).

1. Enumerate all vertex subsets of size \( k \) in \( G \). For each such vertex subset \( S_1 \), enumerate all vertex subsets of size \( k \) in \( H \);

2. for each such subset \( S_2 \) in \( H \), we enumerate all possible one to one mappings between \( S_1 \) and \( S_2 \);

3. for each such mapping \( M \), determine whether the subgraph induced by \( S_1 \) in \( G \) is isomorphic to the subgraph induced by \( S_2 \) in \( H \) under \( M \) or not;

4. if there exists a mapping that can make the subgraph induced by \( S_1 \) in \( G \) isomorphic to the subgraph induced by \( S_2 \) in \( H \), call the algorithm in Lemma 4.1 to compute a largest common subgraph of \( G \) and \( H \) and return it;

5. otherwise, for each integer \( i \) between 1 and \( k \), use the same approach as described in steps 1, 2, 3 to determine whether \( G \) and \( H \) contains a common subgraph of size \( i \) or not;

6. Based on the result of the exhaustive search performed in step 5, return a common subgraph of the largest size.

We then show that the algorithm can compute the largest common subgraph of \( G \) and \( H \) in expected \( 2^{O(n^{\frac{1}{2}} \log^{\frac{1}{2}} n)} \) time. In particular, the computation time needed by the exhaustive search performed in steps 1, 2, and 3 is at most

\[
C \left( \binom{n}{k} \binom{m}{k} k! k^2 \right) \leq C n^k m^k \leq C n^{2k} \leq C 2^{2k \log_2 n} = 2^{O(n^{\frac{1}{2}} \log^{\frac{1}{2}} n)}
\]

where \( C \) is some positive constant. The first inequality is due to the fact that \( \binom{m}{k} k! \leq m^k \) and \( \left( \binom{n}{k} k^2 \right) \leq n^k \) for sufficiently large \( n \). From Lemma 4.1 step 4 of the algorithm, if executed, needs \( 2^{hm \log_2 n} \) computation time, where \( h \) is some positive constant independent of \( n \) and \( m \). The computation time needed by step 5 is at most

\[
D \sum_{i=1}^{k-1} \binom{n}{i} \binom{m}{i} i! i^2 \leq D \sum_{i=1}^{k-1} n^i m^i i^2 \leq D (kn)^{2k} k^2 \leq D 4^{3 \log_2 k + 2k \log_2 n} = 2^{O(n^{\frac{1}{2}} \log^{\frac{1}{2}} n)}
\]

where \( D \) is some positive constant. The first inequality is due to the fact that \( \binom{m}{i} i! \leq m^i \) and \( \binom{n}{i} \leq n^i \). The second inequality follows from the fact that \( 1 \leq i < k \).

Only one of steps 4 and 5 is executed by the algorithm. From Lemma 4.2, the probability for step 4 to be executed is at most \( 2^{-\mu n \log^{\frac{1}{2}} n} \), where \( \mu \) is some constant that depends on \( p \) and \( q \) only. The expected computation time to execute steps 4 and 5 is thus at most

\[
2^{O(n^{\frac{1}{2}} \log^{\frac{1}{2}} n)} + 2^{hm \log_2 n} 2^{-\mu n \log^{\frac{1}{2}} n}.
\]

Since \( m \leq n \), the second term is bounded by a constant for sufficiently large \( n \). We thus can conclude that the expected computation time for steps 4 and 5 is \( 2^{O(n^{\frac{1}{2}} \log^{\frac{1}{2}} n)} \). Since steps 1, 2, 3 also need \( 2^{O(n^{\frac{1}{2}} \log^{\frac{1}{2}} n)} \) computation time, the theorem has been proved. \( \square \)
5 Approximate Algorithm

As discussed in the introduction, the maximum independent set problem cannot be approximated
within a ratio of $n^{1-\epsilon}$ in polynomial time unless P=NP, where $\epsilon$ is any positive constant. In [4],
it is shown that the maximum independent set in a graph can be approximated within a ratio of $O\left(\frac{1}{\log_2 n}\right)$. In [15], the approximation ratio is improved to $O\left(\frac{\log \log_2 n}{\log_2 n}\right)$. The result so far remains the best known approximation ratio achieved for this problem in general graphs. In [19, 22, 29], a polynomial time algorithm that can approximate the maximum independent set in a random graph within a constant ratio with high probability is developed and analyzed. However, the approximation ratio of the algorithm is not guaranteed to be constant for all graphs. We show that, the maximum independent set in a random graph can be approximated within a ratio of $2\sqrt{n} \log_2 n$ in expected polynomial time, which is a significant improvement compared with the best known approximate ratio for this problem in general graphs.

Theorem 5.1 Given a random graph $G = (V, p)$ in $n$ vertices where $p$ is a positive constant between 0 and 1, the maximum independent set in $G$ can be approximated within a ratio of $\frac{2n}{2\sqrt{\log_2 n}}$ in expected polynomial time.

Proof. We use the following simple algorithm to compute an independent set in $G$. We let $k = \lceil 2\sqrt{\log_2 n} \rceil$ and partition the vertices in $G$ into $l$ disjoint vertex subsets such that $l - 1$ of them contains $k$ vertices and the remaining one contains at most $k$ vertices. We use $G_1, G_2, \ldots, G_l$ to denote the subgraph induced by vertices in these vertex subsets. It is not difficult to see that $l \leq \left\lfloor \frac{n}{k} \right\rfloor + 1$.

We then use the algorithm we have developed in Theorem 2.1 to compute a maximum independent set in each of $G_1, G_2, \ldots, G_l$ and return the one that contains the largest number of vertices.

We first show that the algorithm returns an independent set in expected polynomial time. $G_1, G_2, \ldots, G_l$ are disjoint and the expected time needed to compute a maximum independent set in each of them is at most $2^{c \log_2 k}$, where $c$ is some positive constant that only depends on $p$. Since $k \leq 2\sqrt{\log_2 n}$, the expected computation time needed to compute the maximum independent set in one subgraph is at most $2^{c \log_2 n} = n^c$. The algorithm thus returns an independent set in expected time $n^{c+1}$.

We then show that the algorithm can achieve an approximate ratio of $\frac{2n}{2\sqrt{\log_2 n}}$. We use $APX(G)$ to denote the size of the independent set returned by the algorithm and $OPT(G)$ to denote the size of a maximum independent set in $G$. We assume that $I$ is a maximum independent set in $G$. Since we have partitioned the graph $G$ into $l$ disjoint subgraphs $G_0, G_1, \ldots, G_l$, at least one of the $l$ subgraphs contains at least $\frac{OPT(G)}{l}$ vertices from $I$. These vertices form an independent set in the subgraph. Since the algorithm computes a maximum independent set in each subgraph and returns the one with the largest size, we immediately obtain

$$APX(G) \geq \frac{OPT(G)}{l}$$

this suggests that

$$\frac{OPT(G)}{APX(G)} \leq l$$

$$\leq \left\lfloor \frac{n}{k} \right\rfloor + 1$$

$$\leq \frac{n}{k} + 1$$

$$\leq \frac{n}{2\sqrt{\log_2 n}} + 1$$

$$\leq \frac{2n}{2\sqrt{\log_2 n}}$$
The second inequality is due to the fact that \( l \leq \lfloor \frac{n}{k} \rfloor + 1 \). The fourth inequality is due to the fact that \( k \geq 2^{\sqrt{\log_2 n}} - 1 \). The last inequality holds for sufficiently large \( n \). The theorem thus has been proved.

6 Conclusions

In this paper, we study the independent set problem in random graphs. We show that a maximum independent set in a random graph can be computed in expected subexponential time. We also show that the parameterized independent set problem is fixed parameter tractable with high probability for random graphs. Using techniques based on enumeration, we show that the largest common subgraph in two random graphs can be computed in expected subexponential time. Our work also suggests that the maximum independent set in a random graph can be approximated within a ratio of \( \frac{2k^{\sqrt{\log_2 n}}}{n} \) in expected polynomial time, which significantly improves on the best known approximate ratio for this problem in general graphs.

It remains unknown whether the maximum independent set in a random graph can be computed in expected polynomial time or not. One possible direction of future work is to study whether there exists such an algorithm. Another related open question is that if such an algorithm does not exist, whether it can be approximated within an improved ratio in expected polynomial time. Further investigations are needed to solve these problems.

References

[1] F. N. Abu-Khzam, N. F. Samatova, M. A. Rizk, and M. A. Langston, “The Maximum Common Subgraph Problem: Faster Solutions via Vertex Cover”, Proceedings of 2007 IEEE/ACS International Conference on Computer Systems and Applications (AICCSA 2007), pp. 367-373, 2007.

[2] E. Balas and C. S. Yu, “Finding a Maximum Clique in An Arbitrary Graph”, SIAM Journal on Computing, 15(4):1054-1068, 1986.

[3] R. Battiti and M. Protasi, ”Reactive Local Search for the Maximum Clique Problem”, Algorithmica 29(4): 610-637, 2001.

[4] R. Boppana and M. Halldörson, “Approximating Maximum Independent Sets by Excluding Subgraphs”, BIT Computer Science and Numerical Mathematics,32(2):180-196, 1994.

[5] K. V. Brinda, S. Vishveshwara and S. Vishveshwara, “Random Network Behaviour of Protein Structures”, Molecular, BioSystems, 6:391-398, 2010.

[6] R. Carraghan and P. M. Pardalos, “An Exact Algorithm for the Maximum Clique Problem”, Operations Research Letters, 9(6): 375-382, 1990.

[7] J. Chen, X. Huang, I. A. Kanj, and G. Xia, “Linear FPT Reductions and Computational Lower Bounds”, Proceedings of the Thirty-Sixth ACM Symposium on Theory of Computing (STOC 2004), pp.212-221, 2004.

[8] J. Chen, X. Huang, I. A. Kanj, and G. Xia, “Strong Computational Lower Bounds via Parameterized Complexity”, Journal of Computer and System Sciences, 72(8):1346-1367, 2006.

[9] I. Dinur and S. Safra, “The Importance of Being Biased”, Proceeding of the Thirty-Fourth ACM Symposium on Theory of Computing (STOC 2002), pp. 33-42, 2002.

[10] R. G. Downey and M. R. Fellows, Parameterized Complexity, Springer-Verlag, 1998.

[11] R. G. Downey and M. R. Fellows, “Fixed Parameter Tractability and Completeness i: Basic Theory”, SIAM Journal of Computing, 24:873-921, 1995.
[12] R. G. Downey and M. R. Fellows, “Fixed Parameter Tractability and Completeness ii: Completeness for W[1]”, Theoretical Computer Science A, 141:109-131, 1995.

[13] P. Erdős and A. Rényi, “On Random Graphs”, Publicationes Mathematicae, 6: 290-297, 1959.

[14] T. Fahle, “Simple and Fast: Improving a Branch-And-Bound Algorithm for Maximum Clique”, Proceedings of the Tenth European Symposium on Algorithms pp. 47-86, 2002.

[15] U. Fiege, “Approximating Maximum Clique by Removing Subgraphs”, SIAM Journal on Discrete Mathematics, 18(2):219-225, 2004.

[16] U. Fiege and E. Ofek, “Finding A Maximum Independent Set in A Sparse Random Graph”, SIAM Journal on Discrete Mathematics, 22(2):693-718, 2008.

[17] F. V. Fomin, F. Grandoni, and D. Kratsch, “Measure and Conquer: A simple $O(2^{0.288n})$ Independent Set Problem”, Proceedings of the Seventeenth ACM-SIAM Symposium on Discrete Algorithms (SODA 2006), pp. 18-25, 2006.

[18] M. R. Garey and D. S. Johnson, Computers and Intractability, W. H. Freeman and Co., San Francisco, California, 1979. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences.

[19] G. R. Grimmett and C. J. H. Medalism, “On Colouring Random Graphs”, Mathematical Proceedings of the Cambridge Philosophical Society, 77(2):313-324, 1975.

[20] A. Grosso, M. Locatelli, F. D. Croce, “Combining Swaps and Node Weights in An Adaptive Greedy Approach for the Maximum Clique Problem”, Journal of Heuristics 10(2):135-152, 2004.

[21] J. Hästad, “Clique Is Hard to Approximate Within $n^{1-\epsilon}$”, Proceedings of the 37th Annual Symposium on Foundations of Computer Science (STOC 1996), 627-636, 1996.

[22] S. Homer and M. Peinado, “On the performance of Polynomial-time CLIQUE Approximation Algorithms on Very Large Graphs”, In Cliques, Coloring, and Satisfiability: second DIMACS Implementation Challenge, pp. 103-124, 1993.

[23] T. Jian, “An $O(2^{0.308n})$ Algorithm for Solving Maximum Independent Set Problem”, IEEE Transactions on Computers, 35(9):847-851, 1986.

[24] D. S. Johnson, “Approximate Algorithms for Combinatorial Problems”, Journal of Computer and System Sciences, 9, 256-278, 1974.

[25] R. M. Karp, “The Probability Analysis of Some Combinatorial Search Problems”, Algorithms and Complexity: New Directions and Recent Results, 1-19, Academic Press, New York, 1976.

[26] K. Katayama, A. Hamamoto, and H. Narihisa, “An Effective Local Search for the Maximum Clique Problem”, Information Processing Letters 95(5):503-511, 2005.

[27] J. Konc and D. Janežič, “An Improved Branch and Bound Algorithm for the Maximum Clique Problem”, MATCH Communications in Mathematical and in Computer Chemistry 58(3): 569-590, 2007.

[28] O. Kuchaiev, T. Milenković, V. Memišević, W. Hayes, and N. Pržulj, “Topological Network Alignment Uncovers Biological Function and Phylogeny”, Journal of Royal Society Interface, 7(50):1341-1354, 2010.

[29] A. Coja-Oghlan and C. Efthymiou, “On Independent Sets in Random Graphs”, Proceedings of the Twenty Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2011), pp. 136-144, 2011.

[30] P. R. J. Östergård, “A Fast Algorithm for the Maximum Clique Problem”, Discrete Applied Mathematics 120 (13):197-207, 2002.
[31] P. M. Pardalos and G. P. Rogers, “A Branch and Bound Algorithm for the Maximum Clique Problem”, *Computers and Operations Research* 19 (5): 363-375, 1992.

[32] J. C. Régis, “Using Constraint Programming to Solve the Maximum Clique Problem”, *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming*, pp. 634-648, 2003.

[33] J. M. Robson, “Algorithms for Maximum Independent Sets”, *Journal of Algorithms*, 7(3):425-440, 1986.

[34] J. M. Robson, “Finding A Maximum Independent Set in Time $O(2^\frac{n}{4})$”, *Technical Report 1251-01*, LaBRI Université de Bordeaux I, 2001.

[35] W. H. Suters, F. N. Abu-Khzam, Y. Zhang, C. T. Symons, N. F. Samatova, and M. A. Langston, “A New Approach and Faster Exact Methods for the Maximum Common Subgraph Problem”, *Proceedings of the Eleventh International Computing and Combinatorics Conference*, pp. 717-727, 2005.

[36] R. E. Tarjan and A. E. Trojanowski, “Finding A Maximum Independent Set”, *Technical Report CS-TR-76-550*, Stanford University, 1976.

[37] E. Tomita and T. Seki, “An Efficient Branch-and-bound Algorithm for Finding a Maximum Clique”, *Discrete Mathematics and Theoretical Computer Science*, pp. 278-289, 2003.

[38] E. Tomita and T. Kameda, “An Efficient Branch-and-bound Algorithm for Finding A Maximum Clique with Computational Experiments”, *Journal of Global Optimization* 37(1): 95-111, 2007.