Towards Maximum Independent Sets on Massive Graphs

Yu Liu[†] Jiaheng Lu[§] Hua Yang[†] Xiaokui Xiao[‡] Zhewei Wei[†]

[†]*DEKE, MOE and School of Information, Renmin University of China*
[§] *Department of Computer Science, University of Helsinki, Finland*
[‡]*School of Computer Engineering, Nanyang Technological University, Singapore*

ABSTRACT

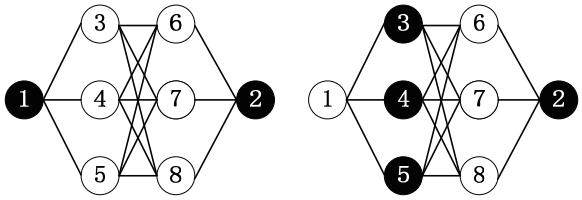
Maximum independent set (MIS) is a fundamental problem in graph theory and it has important applications in many areas such as social network analysis, graphical information systems and coding theory. The problem is NP-hard, and there has been numerous studies on its approximate solutions. While successful to a certain degree, the existing methods require memory space at least linear in the size of the input graph. This has become a serious concern in view of the massive volume of today's fast-growing graphs.

In this paper, we study the MIS problem under the *semi-external* setting, which assumes that the main memory can accommodate all vertices of the graph but not all edges. We present a greedy algorithm and a general *vertex-swap* framework, which swaps vertices to incrementally increase the size of independent sets. Our solutions require only few sequential scans of graphs on the disk file, thus enabling in-memory computation without costly random disk accesses. Experiments on large-scale datasets show that our solutions are able to compute a large independent set for a massive graph with 59 million vertices and 151 million edges using a commodity machine, with a memory cost of 469MB and a time cost of three minutes, while yielding an approximation ratio that is around 99% of the theoretical optimum.

1. INTRODUCTION

The *maximum independent set (MIS)* problem is a longstanding problem in graph theory. An independent set is a set of vertices in a graph, such that no two vertices in the set are connected by an edge. That is, each edge in the graph has at most one endpoint in the set. A maximal independent set is an independent set such that adding any other vertex to the set forces the set to contain an edge. A graph may have many maximal independent sets of different sizes; the largest maximal independent set is referred to as the *maximum independent set*, and its size is referred to as the *independence number*. For example, in Figure 1, $\{v_1, v_2\}$ is a maximal independent set, while $\{v_2, v_3, v_4, v_5\}$ is the maximum independent set, and the independence number of the graph equals four.

The MIS problem is closely related to a number of fundamental graph problems [5, 11, 14, 20], such as maximum common induced subgraphs, minimum vertex covers, graph coloring, and maximum common edge subgraphs, etc. Its significance is not just limited to graph theory but also in numerous real-world applications, such as indexing techniques for shortest path and distance queries [11, 17], automated labeling of maps [22], information coding [9], signal transmission analysis [24], social network analysis [13], and computer vision [27]. In particular, the state-of-the-art techniques [11, 17] for shortest path and distance queries require building some graph indexing structures, the construction of which requires repeatedly invoking a sub-routine for solving the MIS problem. In addition, in the *map labeling problem* (which seeks to accurately assign a set of labels to various regions in a map), a major challenge is to avoid assigning conflicting labels to the same region; this challenge can be solved by computing the MIS of an *intersection graph* [22] constructed based on the map and the label constraints.



(a) Maximal independent set (b) Maximum independent set

Figure 1: An example to illustrate that $\{v_1, v_2\}$ is a maximal independent set, but $\{v_2, v_3, v_4, v_5\}$ is a maximum independent set.

The MIS problem is known to be NP-hard, and does not admit a constant factor approximation (for general graphs) [14, 20]. In particular, Hastad [16] shows that for any $\varepsilon > 0$, there is no polynomial-time $n^{1-\varepsilon}$ approximation algorithm for the maximum independent set problem, unless NP=ZPP. Halldórsson and Radhakrishnan [14] show that for general graphs of bounded degree Δ, the greedy algorithm produces $\frac{\Delta+1}{3}$-factor approximation result for the MIS problem. The approximation factor can be improved to $\frac{2\bar{d}+3}{5}$ using the fractional relaxation technique of Nemhauser and Trotter [19], where $\bar{d}$ is the average degree of the graph. The best approximation ratio known for the MIS problem is $O(n(\log\log n)^2/(\log n)^3)$ [10]. In addition, there are several studies on exponential-time exact algorithms: Robson [20] solves the problem in time $O(2^{0.276n})$ using exponential space, which is recently improved to $O(1.2002^n \cdot n^{O(1)})$ by Xiao [26].

All the above algorithms require memory space linear in the size of the input graph. This requirement, however, is often unrealistic for the large graphs (e.g., social networks, web graphs) commonly seen in modern applications. To address this issue, in this paper, we study I/O efficient *semi-external* algorithms for the MIS problem.

Proceedings of the VLDB Endowment, Vol. 8, No. 13

Methods@	I/O or CPU bound	Approx. Ratio
Xiao [26]	CPU: O($1.2002^{\|V\|} \cdot \|V\|^{O(1)}$)	Optimal
Halldórsson [14]	CPU: O($\|V\|\log\|V\|+\|E\|$)	$(2\bar{d}+3)/5$
Zeh [27]	I/O: O(sort($\|E\|+\|V\|$))	No bound
Greedy	I/O: $\frac{\|V\|+\|E\|}{B}(\log_{\frac{M}{B}} \frac{\|V\|}{B} + 2)$	Proposition 2
One-k-swap	I/O: O(scan($\|V\| + \|E\|$))	Proposition 5
Two-k-swap	I/O: O(scan($\|V\| + \|E\|$))	Better

Table 1: Time and I/O cost and performance ($\bar{d}$: the average degree)

In particular, we assume that the main memory can accommodate all vertices, but not all edges, of the input graph. This assumption is valid even for a massive graph with one hundred million vertices. In particular, assume that we use a 4-byte integer to represent one vertex ID, the memory cost for storing all vertices is only 0.4G bytes, which is much smaller than the memory size of a commodity machine nowadays.

1.1 Main results

This paper makes several contributions as follows. First, we develop a semi-external greedy algorithm for the MIS problem, which constructs an independent set by performing *one* sequential scan of the disk file of the input graph, avoiding expensive random disk accesses. This algorithm is simple, and yet, as our empirical experiments show, it yields a good approximate ratio against most real data sets.

Second, we develop a *one-k-swap* algorithm, which takes the independent set computed from our greedy algorithm and swaps vertices to enlarge the independent set. Although similar ideas of node-swap are mentioned in previous work (e.g., [18]), we are facing a new challenge under the *semi-external* setting: we need to guarantee the completeness and correctness of the swaps by relying on only sequential scans of disk files with a limited amount of main memory. Towards addressing the challenge, we develop a mechanism to solve a problem called "*swap conflict*", which ensures that all one-k-swaps are correctly performed with only few sequential scans of adjacent files of graphs.

Third, we propose a *two-k-swap* algorithm, which exchanges two vertices in the independent set with other k ($k \geq 3$) vertices that are not in the independent set. *Two-k-swap* supports a wider range of swaps, and thus, further increases the size of the independent set. Table 1 shows a summary on computing complexity and performance of different approaches for maximum independent sets.

Fourth, we provide in-depth theoretical analysis of our algorithms under the well-adopted *Power-Law Random (PLR)* graph model [3]. We show that, under mild assumptions on the input graph, our algorithms can return a solution with an expected approximation ratio of 99%. Furthermore, this performance can be achieved with a limited number of sequential scans, which is highly desirable under the semi-external setting.

Finally, we experimentally evaluate our algorithms using several large-scale real and synthetic datasets. For example, to process the Facebook graph with 59 million vertices (1.57GB in disk), our two-k-swap algorithm requires only 469M bytes of the main memory to achieve the 99% approximation ratio. For the Twitter graph with 2.4 billion edges (9.41GB), the two-k-swap algorithm consumes 524MB main memory and less than an hour, while achieving the 97% approximation ratio. We also made experiments for a large Clueweb12 data with 42 billion edges (169GB), which requires 5.7GB in the main memory to obtain the 97% ratio. Therefore, to our knowledge, this is the first comprehensive result in the literature that demonstrates memory-efficient and time-efficient algorithms for the MIS problem on billion-edge real graphs.

The remainder of this paper is organized as follows. Section 2 discusses the preliminaries and the problem statement. Section 3 provides the related work. Section 4 describes the greedy approach and we present the one-k-swap and two-k-swap algorithms in Section 5 and Section 6 respectively. We report our experimental results in Section 7 and conclude this paper in Section 8.

2. PRELIMINARIES

In this section, we first formulate the research problem, and then introduce the Power-Law Random (PLR) graph model, which will be used to analyze the performance of our algorithms later.

2.1 Problem Statement

Let $G = (V, E)$ be a simple undirected graph with vertex set V and edge set E. The neighborhood of a vertex $v \in V$ (i.e., the set of vertices adjacent to v) is denoted $N(v)$, and the degree of v (i.e., the number of its neighbors) $deg(v)$. We use the adjacent lists to represent a graph and sort the adjacent list of each vertex in the ascending order of vertex degrees.

An independent set in G is a subset of pairwise non-adjacent vertices. A maximal independent set is an independent set such that any vertex not in the set is adjacent to at least one vertex in the set. A maximum independent set is an independent set of the largest possible size for a given graph G. This size is referred to as the independence number of G. The problem that we study is defined as follows.

Problem Statement: Compute an independent set as large as possible for an undirected graph $G(V, E)$ with limited memory M. Here, $c|V| \leq M \ll |G|$, where c is a small constant number (e.g., c = 2 or 3), $|V|$ is the number of vertices, and $|G|$ is the space for the entire graph G.

2.2 Power-Law Random Graph Model

Considerable research has focused on discovering and modeling the topological properties of various large-scale real-world graphs, including social graphs and web graphs. In what follows, we consider a power-law random graph P(α, β) [3] with the following degree distribution parameterized with two given values α and β: For any x, the number of vertices in G with degree x equals y, such that log$y$ = α - βlog$x$. In other words, we have

$$|v : deg(v) = x| = y = \frac{e^{\alpha}}{x^{\beta}} \quad (1)$$

It can be verified that α is the logarithm of the size of the graph and β can be regarded as the log-log growth rate of the graph. Given a degree sequence for each vertex v, we use the following three steps to construct a random graph model: (1) Form a set L containing $deg(v)$ distinct copies of each vertex v; (2) Choose a random matching of the elements of L; (3) For two vertices u and v, the number of edges joining u and v is equal to the number of edges in the matching of L joining copies of u to copies of v. This random graph model will be used in the following sections to investigate the expected approximate ratio of various algorithms.

3. RELATED WORK

As mentioned in Section 1, the MIS problem is NP-hard and does not admit a constant factor approximation for general graphs. Furthermore, Shen et al. [21] shows that the problem is APX-hard even for power-law random graphs. In this section, we first review the existing in-memory algorithms for the MIS problem, and then discuss related work on external algorithms.

In-memory algorithms. A brute-force algorithm requires $O(2^n)$ time by examining every vertex subset and checking whether it is an independent set. The complexity is reduced to $O(2^{0.276n})$ by Robson [20] with a modified recursive algorithm. This result is recently improved by Xiao et al. [26] to $O(1.2002^n \cdot n^{O(1)})$. These exact methods are applicable to problem instances of very limited sizes. For larger cases, various heuristics have been proposed. The most representative heuristics include tabu search [24], stochastic local search [4], simulated annealing [12], variable neighborhood search [15], and evolutionary algorithms [8].

Approximating the MIS problem with theoretical bounds has been extensively studied [5]. Wei [25] presents an algorithm which can produce an $\sum_{v \in V} \frac{1}{deg(v)+1}$ approximation. Halldórsson and Radhakrishnan [14] show that for general graphs of bounded degree Δ, the greedy algorithm produces a $\frac{\Delta+1}{3}$-factor approximation. The approximation factor can be improved to $\frac{2\bar{d}+3}{5}$ using the fractional relaxation technique of Nemhauser and Trotter [19], where $\bar{d}$ is the average degree of the graph. In terms of the number of vertices in graphs, Feige shows the currently known best approximation ratio: O(n(loglog$n$)^2/(log$n$)^3) [10].

In general, existing approximate solutions are more efficient than the exact methods, but they assume that the whole graph always fits in the main memory, which may not be true for the current fast-growing graphs such as social graphs often containing billions of edges. In this paper, we present algorithms that are scalable to graphs with 42 billion edges using a single commodity PC.

External algorithms. I/O-efficient graph algorithms have received considerable attention because massive graphs arise naturally in many applications. Vitter [23] presents an excellent survey of the state-of-the-art algorithms and data structures for external memory. Although there are a vast number of studies for maximum independent set in memory, very few of them study I/O efficient algorithms. In particular, Abello et al. [2] propose randomized algorithms for the problem of maximal independent set; their I/O-complexity is $O(sort(|E|))$ with high probability. Zeh [27] proposes the first deterministic external algorithm for the maximal independent set problem, which is implemented with time-forward processing using an external priority queue. The I/O-complexity is $O(sort(|V|+|E|))$. Note that these algorithms all focus on the maximal independent set problem, and they cannot provide any guarantee on the performance of *maximum* independent set. To the best of our knowledge, we present the first comprehensive study to bridge the theory and practice in *maximum* independent sets with the semi-external setting for massive graphs.

4. GREEDY ALGORITHM

In this section, we propose a semi-external greedy algorithm for maximum independent sets and analyze its performance ratio based on the PLR graphs introduced in Section 2.2.

4.1 Algorithm Description

Algorithm 1 presents the pseudo-code for a greedy algorithm following the semi-external model. The high-level idea is to repeatedly add vertices with small degrees to the independent set IS if none of their neighbours is added to IS, until no more vertices can be added. In particular, given a sorted adjacent file by the ascending order of the degree of vertex, the algorithm first initializes the states of all vertices to be *INITIAL* (Lines 1-2), which means that these vertices ares still unvisited. Then Lines 3-8 update the states of its neighbours until all vertices are visited, where $N(u)$ denotes the set of vertices adjacent to u. Finally, a maximal independent set is returned for all IS vertices.

Algorithm 1 needs a preprocessing phase to sort the vertices by degrees and then scans the adjacent lists only once. Note that if we were to sort $|V|+|E|$ keys in external memory, the I/O cost would be $sort(|V|+|E|) = \frac{|V|+|E|}{B} \log_{\frac{M}{B}} \frac{|V|+|E|}{B}$, where B is the block size and M is the main memory size. However, since there are only $|V|$ adjacent lists and each list fits in the main memory based on the semi-external model, we can reduce the number of passes in the sorting algorithm with the following partition scheme. We first make a scan to partition the adjacent lists into groups, each of size B. Then we sort the adjacent lists in each group with one pass[1], and run external memory sorting algorithm on the $|V|/B$ groups. This will reduce the number of passes to $\log_{\frac{M}{B}} \frac{|V|}{B}$, and thus the I/O cost is $\frac{|V|+|E|}{B}(\log_{\frac{M}{B}} \frac{|V|}{B} + 1)$. Finally, the scan cost is $\frac{|V|+|E|}{B}$. Therefore, the total IO cost is $\frac{|V|+|E|}{B}(\log_{\frac{M}{B}} \frac{|V|}{B} + 2)$.

Algorithm 1: Semi-external Greedy Algorithm

Input: A sorted adjacent-list file for graph G
Output: A maximal independent set of G

```
1 for v ∈ V of G do
2     State[v] ← INITIAL;
3 for v ∈ V of G do
4     if State[v]=INITIAL then
5         State[v] ← IS;
6         for each u in N(v) do
7             if State[u]=INITIAL then
8                 State[u] ← ∼IS;
9 Return all vertices whose states are IS;
```

Remark. We compare the above algorithm with the existing in-memory greedy algorithm [14], called DynamicUpdate hereafter. The main difference between DynamicUpdate and the above algorithm is that if any vertex v is added to the independent set, then DynamicUpdate needs to remove v and its neighbors from the graph, and dynamically update the degrees of affected vertices, iterating the remain graph until empty. Therefore, DynamicUpdate would incur the frequent random accesses to update the degrees of vertices in the semi-external setting. In the above greedy algorithm, however, we adopt a lazy strategy to avoid expensive random accesses. As shown in the empirical results of Section 7, unlike DynamicUpdate, our greedy algorithm is scalable to massive graphs.

4.2 Performance Analysis

We analyze the approximation ratio of Algorithm 1 in the following paragraphs based on the random P(α, β) graphs. According to Equation (1) in Section 2.2, vertices and edges of the graph can be calculated by the following formula:

$$\begin{aligned} |V| &= \sum_{x=1}^{\Delta} \frac{e^\alpha}{x^\beta} = \zeta(\beta, \Delta)e^\alpha, \\ |E| &= \sum_{x=1}^{\Delta} \frac{e^\alpha}{x^{\beta-1}} = \zeta(\beta-1, \Delta)e^\alpha, \end{aligned} \tag{2}$$

where $\zeta(x, y) = \sum_{i=1}^{y} \frac{1}{i^x}$ and $\Delta = \lfloor e^{\frac{\alpha}{\beta}} \rfloor$ is the maximum degree of the graph. Intuitively, the greedy algorithm prefers to select vertices with small degrees, and thus most of vertices with degree 1 are added to IS, and some of vertices with degree 2, 3 and so on. In this way, Lemma 1 quantifies the expected portion of vertices for the degree i which can be added to IS.

[1]This sorting can be completed with only one pass, because we assume $M \geq B^2$, which is known as the tall-cache assumption [7].

LEMMA 1. *Given a power law graph $P(\alpha, \beta)$, the expected number of independent vertices with degree i returned by Algorithm 1, denoted as $GR_i(\alpha, \beta)$, is*

$$GR_i(\alpha, \beta) \geq \sum_{x=1}^{\lfloor \frac{e^\alpha}{i^\beta} \rfloor} \left(\frac{\frac{ix}{e^\alpha} + \zeta(\beta - 1, \Delta) - \zeta(\beta - 1, i)}{\zeta(\beta - 1, \Delta)} \right)^i \quad (3)$$

where $\Delta = \lfloor e^{\frac{\alpha}{\beta}} \rfloor$ is the maximum degree of the graph.

The proof can be found in the appendix. Based on Lemma 1, summing up for all degrees from 1 to Δ immediately implies the expected number of the independent set.

PROPOSITION 2. *Given a power law graph $P(\alpha, \beta)$, the expected size of the independent set, denoted as $GR(\alpha, \beta)$, returned by the Greedy algorithm, can be computed as follows*

$$GR(\alpha, \beta) = \sum_{i=1}^{\Delta} GR_i(\alpha, \beta) \quad (4)$$

β	1.7	1.8	1.9	2.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7
ratio	0.987	0.986	0.987	0.983	0.983	0.984	0.986	0.986	0.986	0.988	0.988

Table 2: Performance ratios of Algorithm 1 with varied β

Remark. Table 2 demonstrates the performance ratio of the greedy algorithm, which varies parameter β from 1.7 to 2.7 and fixes the number of vertices to 10 million to simulate a massive graph. Note that we now cannot obtain the exact independent number if NP $\neq$ P. Therefore we develop an algorithm to calculate its theoretical upper bound. The detail of our algorithm is shown in the Appendix (see Algorithm 5), which is modified for the semi-external model from the approach in [13]. In our implementation, for each β value, we generate ten random graphs and compute their optimal bounds with Algorithm 5. Then the average number is computed as the final independence number. The size of the independent set from the greedy algorithm is computed by Proposition 2. It can be seen from Table 2 that the ratios are always greater than 0.98, that means the greedy one is a simple yet delightful algorithm, which can return a large independent set with 98.3%-98.8% of the optimum.

5. ONE-K-SWAP ALGORITHM

In this section, we propose a new algorithm, called one-k-swap, which finds a larger independent set than the greedy algorithm. In particular, we first characterize the challenges to perform the swap operations based on the semi-external framework. Then we introduce our algorithm and analyze its effectiveness and efficiency.

5.1 Intuition

In this work, one of our key observations is that we can obtain a larger independent set through *swap* operations. For example, recall Figure 1. Assume that the independent set includes vertices $\{v_1, v_2\}$ in Figure 1(a). Then we can exchange v_1 with v_3, v_4, v_5 to increase the size of the independent set. In the literature, Khanna et al. [18] proposed two types of swaps called $0 \leftrightarrow 1$ swap and $1 \leftrightarrow 2$ swap. $1 \leftrightarrow 2$ swap means to exchange one independent-set (IS) vertex with two non-IS vertices, and $0 \leftrightarrow 1$ swap adds a new IS vertex. It turns out that any $1 \leftrightarrow k$ ($k \geq 3$) swap is equivalent to a combination of one $1 \leftrightarrow 2$ swap and some $0 \leftrightarrow 1$ swaps. Swap is conceptually simple, but there are two technical challenges to design a swap-based algorithm based on the semi-external model:

(1) **Costly random accesses**. The first challenge is *expensive random accesses* to decide whether a swap operation can be performed correctly to guarantee an independent set. To understand this, see Figure 1 again. In order to decide whether v_1 can be exchanged by v_3, v_4 and v_5, one has to access the adjacent lists of those vertices to ensure that there is no edge among v_3, v_4 and v_5. Otherwise, this $1 \leftrightarrow 3$ swap cannot be performed.

(2) **Swap conflict**. The second challenge is *swap conflict*. We use the example in Figure 2 to illustrate it. Assume that v_1 and v_4 are in the independent set. Then v_1 can be exchanged by v_2 and v_3; and v_4 can be exchanged by v_5 and v_6. But these two swaps cannot be performed simultaneously. That means, they *conflict* with each other. Therefore, we need a mechanism to allow one and only one swap to be successfully performed at this example.

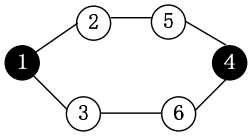


Figure 2: An example to illustrate *swap conflict*.

5.2 Data Structure

Notation	Status	Explanation
I	IS	in the independent set.
N	$Non\text{-}IS$	not in the independent set.
A	$Adjacent$	adjacent to only one IS node.
C	$Conflict$	being non-IS in the next iteration.
P	$Protected$	being IS in the next iteration.
R	$Retrograde$	being non-IS in the next iteration.

Table 3: Summary of the notations for states

To address the above two technical challenges, we define six states for each vertex to record their status during swaps. In particular, an ***IS vertex*** is currently in the independent set. A ***non-IS vertex*** is not in the independent set now. An ***adjacent vertex*** is a non-IS vertex which is adjacent to only one IS vertex. A ***protected vertex*** is an adjacent vertex, which can be swapped to an IS vertex in the next iteration. A ***conflict vertex*** is also an adjacent vertex, but it cannot be swapped to an IS vertex in the next iteration. Finally, a ***retrograde vertex*** is an IS vertex which will be swapped to a non-IS vertex in the next iteration. Therefore, there are six states: $\{I, N, A, P, C, R\}$, as shown in Table 3. In addition, for each adjacent vertex v, we maintain a set ISN(v) to record its IS-Neighbour. Note that the size of an ISN set for each vertex is always one, as each adjacent vertex has only one IS neighbour, For example, see Figure 2 again. ISN(v_2)=ISN(v_3)=v_1, and ISN(v_5)=ISN(v_6)=v_4.

DEFINITION 1. *(1-2 swap skeleton) Given a graph G, we say that there is a 1-2 swap skeleton (u, v, w) if state(u)= state(v)="A", and state(w)="I", ISN(u)= ISN(v)=w, and there is no edge between u and v in G.*

Recall Figure 2. Suppose that the states of v_2, v_3, v_5, v_6 are "A" and those of v_1, v_4 are "I". Then both (v_2,v_3,v_1) and (v_5,v_6,v_4) are 1-2 swap skeletons. Note that a 1-2 swap skeleton is only a valid swap candidate, which is not necessarily swapped in our algorithm. For example, only one of 1-2 swap skeletons between (v_2,v_3,v_1) and (v_5,v_6,v_4) can swap, as they conflict with each other.

Figure 3 shows a state transition diagram. Transition conditions are labeled on the path. We use the following example to illustrate the transition of states.

EXAMPLE 1. *This example provides the intuition how we resolves the above two challenges based on the state transition. See Figure 2 again. Suppose that v_1 and v_4 are in the initial independent set. Assume that the access order of vertices is: v_1, v_4, v_2, v_6, v_3 and v_5. Then we need to scan the file three times (without any random access). In the first scan, state(v_1)=state(v_4)="I" and the states of all other vertices are "A", ISN(v_2)=ISN(v_3)=v_1; ISN(v_5)=ISN(v_6)=v_4. Then in the second scan, the states of v_2 and v_3 are changed to "P" and state(v_1)="R". That means v_2 and v_3 can exchange v_1 in the independent set. Subsequently the state of of v_6 is changed from "A" into "C", because the neighbour of v_6 is v_3 whose state is "P". Intuitively, since v_2 is read before v_6, v_2 has the right of preemption for swap and v_6 cannot swap, which in turn resolves the swap conflict. Finally, in the third scan, the state of v_1 is changed to "N"; and those of v_2 and v_3 become "I". Therefore, a larger independent set v_2, v_3 and v_4 is correctly identified.*

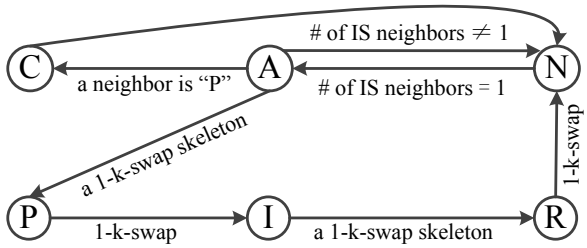


Figure 3: State transition diagram.

5.3 One-k-swap algorithm

Based on the above motivation, we are ready to describe the one-k-swap algorithm. The main driver is clear: it repeatedly scans the adjacent file in the disk and updates the states of vertices in the main memory until no more swaps can be performed. Finally, all vertices with states "I" are returned as an independent set.

We now go through Algorithm 2. First, the vertices that have only one IS neighbour have the potential to swap (Line 1-3). Then the algorithm proceeds in three phases: *pre-swap* (Line 7-14), *swap* (Line 15-19) and *post-swap* (Line 20-28). (1) In the *pre-swap* phase, there are three cases for vertex u: (i) u has conflicted with other swap candidates (Line 9-10); or (ii) there is a new 1-2 swap skeleton for u (Line 11-12); or (iii) there will be a 0-1 swap for u (Line 13-14). (2) In the *swap* phase, more vertices are added to the independent set. (3) Finally, in the *post-swap* phase, some 1-0 swaps are performed to guarantee that the returned result is a maximal set.

EXAMPLE 2. *We use the example in Figure 4 to illustrate the 1-k swap algorithm. Suppose, the initial independent set includes five vertices: v_1, v_4, v_8, v_{12}, v_{14}. Then we run the 1-k swap algorithm to get a larger IS. Figure 4(c) shows the vertex IDs sorted by degrees and Figure 4(d) shows the state transition. In Line 1-3, six vertices have the state "A". Then in Line 11-12, (v_2, v_3, v_1) and (v_7, v_9, v_4) are identified a 1-2 swap skeleton. The states of three vertices v_5, v_6 and v_{10} are changed to "C" due to the swap conflict. In line 22-23, two 1↔2 swaps are performed. Finally, in Line 28, several vertices are labeled as "N" again. Therefore, a larger independent set includes v_2, v_3, v_7, v_8, v_9, v_{12}, v_{14}, shown in Figure 4(b).*

Algorithm 2: One-k-swap Algorithm

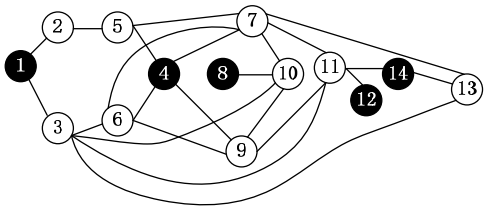
Input: An adjacent-list file of graph G and an initial independent set
Output: A larger independent set of G

```
1  for node u ∈ G do
2      if u has only one adjacent IS vertex, say e then
3          state(u) ← (A); ISN(u) ← e ;
4  canSwap=TRUE;
5  while canSwap do
6      canSwap=FALSE;
7      for node u ∈ G do
8          if state(u)=A then
9              if u has a neighbour with state "P" then
10                 state(u) ← C;
11             else if there is a 1-2 swap skeleton (u, v, w) ∈ G
               then
12                 state(u) ← P; state(w) ← R;
13             else if ISN(u)=w ∧ state(w)=R then
14                 state(u) ← P;
15     for node u ∈ G do
16         if state(u)=P then
17             state(u) ← I;
18         else if state(u)=R then
19             state(u) ← N; canSwap= TRUE;
20     for node u ∈ G do
21         if state(u)=N then
22             if all neighbours of u are "C" or "N" then
23                 state(u) ← I;
24         else if state(u)=C ∨ state(u)=A then
25             if u has only one IS adjacent vertex, say e then
26                 state(u) ← A; ISN(u) ← e ;
27             else
28                 state(u) ← N;
29 Return all vertices whose states are "I";
```

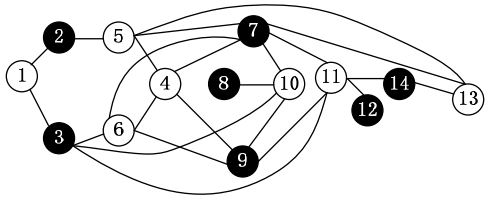
5.4 Performance Analysis

The one-k-swap algorithm is sound and complete in that every swap is performed correctly to maintain an independent set and there exists no vertex which can perform *one-k-swap* to increase the size of the independent set any more. The soundness is because we can always detect the *swap conflict* with the proper state transition and the completeness is because, for every valid 1↔k swap, it must contain a 1-2 swap skeleton.

In the one-k swap algorithm, one round of swap needs three iterations of scan, where in an iteration it needs to access the adjacent file of vertices sequentially. Thus the total number of I/O accesses depends on the number of rounds of swaps. Unfortunately, in the worst-case scenario, the number of rounds is linear to the number of vertices. Figure 5 shows an example, called a *cascade-swap* graph, where only one 1-2 swap can occur in one round of swaps. Therefore, this graph needs three rounds of swaps. That is, $v_7 \rightarrow \{v_8, v_9\}$, $v_4 \rightarrow \{v_5, v_6\}$, and $v_1 \rightarrow \{v_2, v_3\}$. In the worst case, the number of the round of swaps is $n/3$ for a *cascade-swap* graph with n vertices. However, in practice, this worst case occurs rarely. In the following, under the prominent power-law random



(a): The graph before the one-k-swap algorithm.



(b): The graph after the one-k-swap algorithm.

Degree	1	1	2	2	2	4	4	4	4	4	4	4	5	6
ID	8	12	1	2	14	4	5	9	6	10	11	13	3	7

(c): Graph vertices sorted by degrees in ascending order.

ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Initial	I	N	N	I	N	N	N	I	N	N	N	I	N	I
Line 1- 3	I	A	A	I	A	A	A	I	A	A	N	I	A	I
Line 8-14	R	P	P	R	C	C	P	I	P	C	N	I	A	I
Line 15-19	N	I	I	N	C	C	I	I	I	C	N	I	A	I
Line 20-28	N	I	I	N	N	N	I	I	I	N	N	I	N	I

(d): State transitions in the one-k-swap algorithm.

Figure 4: An example graph to illustrate the one-k-swap algorithm.

P(α, β) model, we study the performance of the one-k-swap algorithm through only one round of swap.

To analyze the number of vertices which are added to the independent set in the first round of swap, we first build a lemma to estimate the maximum degree of vertices which can contribute to 1-k swaps. The intuition here is that the degree of some non-IS vertex is too large to become an IS vertex through 1-k swap.

LEMMA 3. *Let d_s denote the maximal degree of vertices which contribute to 1-k swaps. Then $d_s \leq \frac{\alpha+ln\zeta(\beta,e^{\frac{\alpha}{\beta}})}{lnc'(\alpha,\beta)}$ with high probability, where $c(\alpha,\beta) = \frac{\sum_{i=1}^{\Delta} iGR_i(\alpha,\beta)}{e^{\alpha}}$ and $c'(\alpha,\beta) = \frac{\zeta(\beta-1,\Delta)}{\zeta(\beta-1,\Delta)-2c(\alpha,\beta)}$, and Δ is the maximum degree of the graph.*

The following lemma indicates that in the first round of swap, the degrees of new vertices which are added to IS in a 1-k swap are always no less than those removed from IS.

LEMMA 4. *Let $ISN^{-1}(v)$=$\{u|v = ISN(u)\}$. For any vertex v in the independent set of the greedy algorithm, then $deg(v) \leq min\{deg(u)|u \in ISN^{-1}(v)\}$.*

Based on the above lemma, given any 1-2 swap skeleton (u,v,w), the degree of w is no greater than those of u and v. Then we consider three cases: 1) $deg(u)$=$deg(v)$=$deg(w)$; 2) $deg(w)$=$deg(v)$, $deg(u)$>$deg(w)$ or $deg(w)$=$deg(u)$, $deg(v)$>$deg(w)$; 3) $deg(v)$>$deg(w)$, $deg(u)$>$deg(w)$. For each case, we quantify the number of new vertices added into the independent set from degree 2 to d_s by converting them to the "*bins and balls*" problem.

PROPOSITION 5. *Given a power law graph $P(\alpha,\beta)$, in the first round of swap, the expected number of new IS vertices, called Swap Gain (SG), for the one-k swap based on the greedy algorithm can be computed as:*

$$SG(\alpha,\beta) = \sum_{i=2}^{d_s}(T(i,i,i) + \sum_{j=i+1}^{d_s} T(j,i,i) + \sum_{p=i+1}^{d_s}\sum_{q=p}^{d_s} T(p,q,i)) \quad (5)$$

where $T(x,y,i)$ is the number of new vertices added to the IS set by exchanging vertices with degree i to those with degrees x and y. The detailed formula can be found in Equation (15) of Appendix.

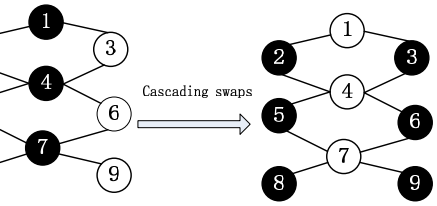


Figure 5: An example to illustrate cascading 1-2 swaps: $v_7 \rightarrow \{v_8, v_9\}$, $v_4 \rightarrow \{v_5, v_6\}$, and $v_1 \rightarrow \{v_2, v_3\}$.

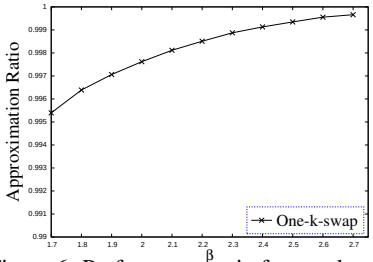


Figure 6: Performance ratio for one-k-swap

Figure 6 shows the performance ratio of the one-round one-k-swap algorithm by varying β from 1.7 to 2.7 and fixing the number of vertices as 10 million again. The size of the independent set is computed by Proposition 5. As shown in Figure 6, the performance ratio is near-optimal $\geq$ 99.5% for all datasets. In addition, compared to Table 2, we observe that this one-round one-k swap is better than the greedy algorithm by a margin 1.5%. Note that this is a noticeable improvement as there is no algorithm that can improve it more than 2% (as the performance of the greedy one is around 98%).

The one-k-swap algorithm needs to scan the adjacent file of the graph multiple times. As shown in theory above and in the experimental results later (Section 7.4), the number of iterations can be limited to a small constant to achieve more than 99.5% performance ratio. Therefore, the I/O cost is $O(scan(|V| + |E|))$ and the memory cost is $2|V|$ for the $state$ array and the *ISN* set.

Finally, to study the time complexity of one-k-swap, the crux is to analyze the costly step in Line 11 that determines if a vertex u involves a *1-k-swap skeleton* (u, v, w). Instead of locating a specific vertex v associated with u, we only need to know if such v exists. Assume that there are x adjacent vertices u' of u, s.t. $ISN(u')$=$ISN(u)$=w, and $state(w)$="I". Let y = $|ISN^{-1}(w)|$. If $y > x$, then there exists at least one vertex v, s.t. ISN(v)=w and there is no edge between u and v. According to Definition 1, (u, v, w) is a 1-k-swap skeleton. In practice, we can reuse the ISN data structure to record the number y for w, as ISN entries for w's are not previously used, and this would not incur any extra main memory cost. With this approach, the time complexity of Line 11 reduces to $O(degree(u))$. Therefore, summing up for all vertices, the total computing cost of one-k-swap is $O(|V| + |E|)$.

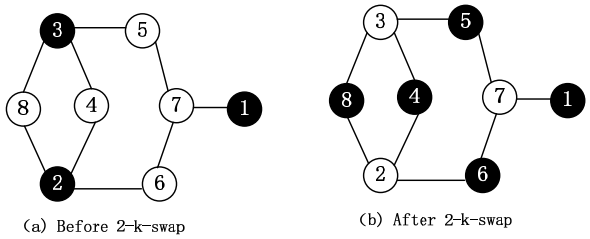
Algorithm 3: Two-k-swap Algorithm

Input: Graph G and an initial independent set
Output: A larger independent set for G

```
1  for node u ∈ G do
2      if u has one or two IS neighbours, say S then
3          state(u) ← A; ISN(u) ← S ;
4  canSwap ← TRUE ;
5  while canSwap do
6      canSwap ← FALSE ;
7      for each vertex u ∈ G do
8          if state(u)=A then
9              Pre-swap(u);
10     for vertex u ∈ G do
11         if state(u)=P then
12             state(u) ← I;
13         else if state(u)=R then
14             state(u) ← N; canSwap ← TRUE;
15     for vertex u ∈ G do
16         if state(u)=C or A or N then
17             if u has one or two IS neighbours, say S then
18                 state(u) ← A; ISN(u) = S ;
19             else
20                 state(u) ← N;
21         if state(u)=N then
22             if all neighbour nodes of u are C or N then
23                 state(u) ← I;
24 Return all nodes whose states are "I";
```

Algorithm 4: Pre-swap(u)

```
1  if ∃v, s.t.(u, v) is a swap candidate for w1, w2 then
2      Add (u, v) to the set SC(w1, w2);
3  if there is an adjacent vertex of u with "P" then
4      state(u) ← C;
5  else if ∃v1, v2, w1, w2 ∈ G, s.t. (v1, v2, u, w1, w2) is a 2-3
   swap skeleton then
6      state(u), state(v1), state(v2) ← P;
7      state(w1), state(w2) ← R;
8      Free the space for SC(w1,w2);
9  else if ∃v, w ∈ G, s.t. (v, u, w) is a 1-2 swap skeleton then
10     state(u) ← P; state(w) ← R;
11 else if ∃u, s.t. ∀w, ISN(u)⊇{w} ∧ state(w)=R then
12     state(u) ← P;
```



(a) Before 2-k-swap (b) After 2-k-swap

Degree	1	2	2	2	2	2	2	3
Vertexid	1	2	3	4	5	6	8	7

(c) Sorted vertices by ascending degrees

Figure 7: An example to illustrate the 2-k-swap algorithm

6. TWO-K-SWAP ALGORITHM

To further enlarge the independent set, a natural step is to extend the one-k-swap algorithm to support *two-k* swaps, which exchanges two IS vertices with three or more non-IS vertices. In the next paragraphs, we describe a new algorithm, called *two-k-swap*.

6.1 Data structure and definitions

Similar to the one-k-swap algorithm, there are still six states in *two-k-swap* for each vertex to present different status, including {I, N, A, P, C, R}. But, the definition of state "A" is changed, where a non-IS vertex may be adjacent to one or *two* IS vertices. Thus, an *ISN* set now may include two IS vertices. We next define *swap candidates* based on *ISN* sets.

DEFINITION 2. *(Swap Candidates) Given a graph G, we say that a vertex pair (u_1, u_2) is a swap candidate for two IS vertices w_1 and w_2, denoted as $(u_1, u_2) \in SC(w_1,w_2)$, if $\forall i$, state(w_i) = "I", state(u_i)="A", ISN(u_i)$\subseteq\{w_1, w_2\}$, and $|ISN(u_1)|$=2, and there is no edge $(u_1, u_2) \in G$.*

DEFINITION 3. *(2-3 swap skeleton) Given a graph G, we say that $(u_1, u_2, u_3, w_1, w_2)$ is a 2-3 swap skeleton if both (u_1, u_2) and (u_1, u_3) are swap candidates for w_1 and w_2, and there is no edge $(u_2, u_3) \in G$.*

See Figure 7(a). Suppose that the states of v_4, v_5, v_6, v_8 are "A" and those of v_2, v_3 are "I". There are four 2-3 swap skeletons in this figure, that is, (v_4,v_5,v_6,v_2,v_3), (v_4,v_5,v_8,v_2,v_3), (v_4,v_6,v_8,v_2,v_3), and (v_8,v_5,v_6,v_2,v_3). It turns out that, as shown later, our algorithm performs a 2↔4 swap for this example.

6.2 Algorithm Two-k-swap

Algorithm 3 shows the pseudo-code of *two-k-swap*, which incrementally enlarges an independent set by performing 2-k swaps and 1-k swaps, and iterating on the graph until no more swaps can be performed.

In particular, Line 1 to 3 find the vertices with state "A" which are potential to swap. Similar to the one-k-swap algorithm, there are three phases at each round: *Pre-swap* (Line 7-9), *Swap* (Line 10-14) and *Post-swap* (Line 15-23). In the *Pre-swap* phase (see Algorithm 4), Line 1-2 add a vertex pair to the SC set, which will be used to find a 2-3 swap skeleton. Subsequently, there are four cases for any *adjacent* vertex u, (i) u has conflicted with other swap candidates (Line 3-4); (ii) u contributes to a 2-3 swap skeleton (Line 5-8); (iii) u contributes to a 1-2 swap skeleton (Line 9-10); (iv) u contributes to a 0-1 swap (Line 11-12). In the *Swap* phase, we perform the swap by changing the state of vertices. Finally in the *Post-swap* phase, Line 16 to 18 add more vertices with state "A" and Line 21-23 perform 0↔1 swap to guarantee a maximal independent set.

EXAMPLE 3. *We use this example to illustrate the two-k-swap algorithm (see Figure 7). Suppose there are three vertices v_1, v_2 and v_3 in the initial independent set. The access order of vertices is shown in Figure 7(c). Then we go through the algorithm. In Line 1-3, v_4, v_5, v_6, v_7 and v_8 are labeled as "A". Subsequently, after v_4 and v_5 are read, the vertex pair (v_4,v_5) is added into SC(v_2,v_3). Then after v_6 is accessed, (v_4,v_5,v_6,v_2,v_3) is identified as a 2-3-swap skeleton (Line 5-8 in Algorithm 4). The state of v_8 is changed to "P", since ISN(v_8)=$\{v_2,v_3\}$, which are labeled as*

"R" now (Line 11-12 in Algorithm 4). The state of v_7 is changed to "C", because it conflicts with v_5 and v_6. Then in Line 10-14, a $2 \leftrightarrow 4$ swap is performed. Finally, in the Post-swap, the state of v_7 is changed to "N" (Line 20). Therefore, a larger independent set includes v_1, v_4, v_5, v_6 and v_8.

Similar to the one-k-swap algorithm, two-k-swap can stop within limited number of iterations to guarantee the efficiency. Therefore, the I/O cost is $O(scan(|V| + |E|))$. The memory cost for the $state$ array and the *ISN* set is no more than $3|V|$, as each *ISN* set contains at most two vertices. The analysis for the auxiliary structure *SC* is more complicated, because the size of *SC* depends on the number of potential 2-3 swap skeletons during one round of swap. A careful study based on P(α, β) random graphs shows that with high probability, the total number of vertices in SC is no more than $|V| - e^{\alpha}$. Therefore, the total memory cost of two-k-swap is bounded by $4|V| - e^{\alpha}$.

LEMMA 6. *Given a power law graph P(α, β), with high probability, the number of vertices in SC sets in the two-k-swap algorithm is no more than $|V| - e^{\alpha}$.*

Finally, we analyze the time complexity of the two-k-swap algorithm. Conceptually, the most costly steps are Lines 1 and 5 in Algorithm 4. In line 1, the worst-case cost is degree(u)+degree(w_1)+degree(w_2), since for each $v \in ISN(w_1)$ or $ISN(w_2)$, we check if there is an edge between u and v by Definition 2. Further, the cost of Line 5 is degree(u)+$|SC(w_1, w_2)|$. It is easy to see that $|SC(w_1, w_2)| \leq$ degree(w_1)+degree(w_2). Therefore, by summing all vertices, the total cost is $O(|V| + |E|+\sum_i degree(w_i))$, where w_i is any vertex which is added to SC sets. It turns out that, based on P(α, β) random graphs, degree(w_i) = $O(log|V|)$ (whose proof is deferred to the Appendix). Note that the number of w_i is obviously less than $|V|$. Therefore, the total time cost is $O(|V|log|V| + |E|)$.

7. EXPERIMENTAL RESULTS

In this section, we set out to verify the effectiveness and efficiency of various algorithms in our framework. First, we look at the effectiveness of algorithms, that is how large the proposed algorithms can find an independent set. Next, we examine the efficiency and the memory cost of algorithms, that is how fast and what cost to obtain a large independent set. Finally, we evaluate the accuracy of our theoretical analysis based on PLRG model.

In our experiments, we implemented six algorithms: DYNAMICUPDATE, STXXL, BASELINE, GREEDY, ONE-K-SWAP and TWO-K-SWAP, where DYNAMICUPDATE is an in-memory greedy algorithm [14] described in Section 4.1; STXXL is an external algorithm [23, 27] based on the open source library STXXL [1]; and BASELINE is similar to GREEDY (Algorithm 1), but without having a global ordering of the vertices by degrees. Among these six algorithms, DYNAMICUPDATE, STXXL, and BASELINE are three existing methods and GREEDY, ONE-K-SWAP and TWO-K-SWAP are three new algorithms proposed in this paper.

7.1 Datasets and Environment

Table 4 shows ten real-life datasets in our experiments. These datasets [6] differ from each other in terms of graph-size, data-complexity, average-degree and degree-distribution. Our goal in choosing these diverse sources is to understand the effectiveness and efficiency of our algorithms in different real environments.

Besides the real data sets, we also generated a set of synthetic data sets based on the power-law $P(\alpha,\beta)$ model, which vary β from 1.7 to 2.7 and fix the number of vertices to 10 million to simulate the real life data sets [3, 21].

Data Set	$\|V\|$	$\|E\|$	Avg. Deg	Disk Size
Astroph	37K	396K	21.1	3.3MB
DBLP	425K	1.05M	4.92	11.2MB
Youtube	1.16M	2.99M	5.16	31.6MB
Patent	3.77M	16.52M	8.76	154MB
Blog	4.04M	34.68M	17.18	295MB
Citeseerx	6.54M	15.01M	4.6	164MB
Uniport	6.97M	15.98M	4.59	175MB
Facebook	59.22M	151.74M	5.12	1.57GB
Twitter	61.58M	2405M	78.12	9.41GB
Clueweb12	978.4M	42.57G	87.03	169GB

Table 4: Datasets and their characteristics

All the algorithms were implemented in C++ and the experiments were performed on a Core i5 CPU 4.0GHz running Windows 7 operating system with 8GB RAM and a 500GB hard disk.

7.2 Independent set size of various algorithms

The first set of experiments was performed to test the effectiveness of various algorithms to find a large independent set. Table 5 shows the results of six algorithms, including DYNAMICUPDATE, STXXL, BASELINE, GREEDY, ONE-K-SWAP and TWO-K-SWAP. We have the following observations: (1) Due to the swap operations, TWO-K-SWAP and ONE-K-SWAP significantly increase the sizes of the independent sets over BASELINE and GREEDY. For example, consider Facebook data with 151 million edges, the sizes of the independent sets from ONE-K-SWAP and TWO-K-SWAP are more than three times larger than those of other algorithms. (2) GREEDY returns a larger independent set than BASELINE for most data sets, thanks to the pre-process sorting phase by degrees; (3) DYNAMICUPDATE is not scalable to large data sets, because it is an in-memory algorithm. But for small data sets which fit in the main memory, DYNAMICUPDATE returns a larger independent set than GREEDY. (4) STXXL is an external algorithm, which can process a large graph. But ONE-K-SWAP and TWO-K-SWAP have much better performance than STXXL. For example, consider the largest Clueweb12 data, ONE-K-SWAP and TWO-K-SWAP return more than 700 million independent vertices, but STXXL finds less than 500 million ones.

Performance ratio We then sought to analyze how the performance of algorithm is compared to the optimum. Figure 8 shows the performance ratios of our three algorithms bases on the synthetic data sets. As we can see, all three algorithms achieve good ratios, and ONE-K-SWAP and TWO-K-SWAP are even better than GREEDY. With the increase of β, the ratio grows. This is expected, as bigger β, less edges in the graph, and better performance for all algorithms. Figure 9 compares the results of TWO-K-SWAP to the optimal bound for real data sets. We observe the similar trend that, for most data sets, such as Facebook, CiteSeerx and Uniport, the size of independent set returned from TWO-K-SWAP reaches 99% that of the optimal bound. Therefore, in practice our algorithms can achieve near-optimal results.

7.3 Running time and memory cost

Now that we have established that the semi-external algorithms proposed in this paper are a good approximation for the maximum independent sets, we turn to evaluate how fast we can obtain such large independent sets for massive graphs.

Data Set	DynamicUpdate	Baseline /STXXL	One-k-swap (after Baseline)	Two-k-swap (after Baseline)	Greedy	One-k-swap (after Greedy)	Two-k-swap (after Greedy)
Astroph	17,948	18,772	18,972	19,036	15,439	16,954	16,970
DBLP	260,984	218,344	258,850	259,198	260,872	273,853	273,853
Youtube	880,876	760,318	865,810	877,905	877,905	881,948	881,962
Patent	2,073,042	1,964,735	2,023,396	2,107,487	2,024,859	2,085,404	2,086,982
Blog	2,116,524	1,693,937	2,004,349	2,063,290	2,094,881	2,151,552	2,151,578
Citeseerx	5,750,794	5,711,727	5,747,513	5,749,859	5,726,927	5,749,983	5,750,026
Uniport	6,947,630	5,840,371	6,932,723	6,938,038	6,943,512	6,947,592	6,947,593
Facebook	N/A	18,893,989	57,269,875	57,986,375	58,226,290	58,232,256	58,232,269
Twitter	N/A	36,072,163	46,978,395	48,059,663	48,121,173	48,742,356	48,742,573
Clueweb12	N/A	499,444,213	703,485,927	725,810,643	606,465,512	723,673,169	729,594,728

Table 5: Number of vertices in the independent sets return by various algorithms

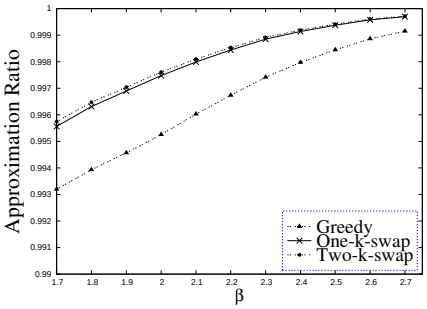


Figure 8: Performance ratio of three algorithms

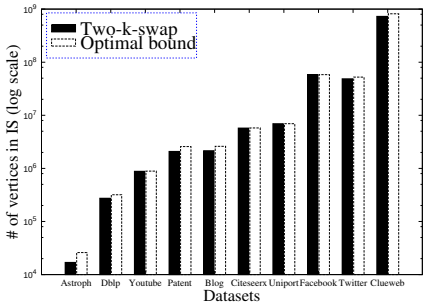


Figure 9: Two-k-swap and the optimal bound.

Table 6 investigates the computing time and the memory cost of each algorithm using different data sets. For most data sets, our three algorithms complete within few minutes. For example, there are 151 million edges and 59 million vertices in Facebook data, the ONE-K-SWAP and TWO-K-SWAP take less than 3 minutes. Consider the biggest Clueweb12 data, which has 42 billion edges and 978 million nodes, ONE-K-SWAP and TWO-K-SWAP consume 8.8 hours and 10.4 hours respectively. This is reasonable since even one sequential scan of such big graph takes more than one hour.

Table 6 also shows the memory consumption of each algorithm, including the memory cost for the auxiliary structures: ISN and SC sets. For example, the Twitter graph is 9.41GB in the disk, but the memory consumption in our approach is limited to 524M. This demonstrates the memory-efficiency of our approach. Figure 10 gives a deep study on the ratio between the maximal number of vertices in SC (for the TWO-K-SWAP algorithm) and the total number of vertices. The ratios are relatively stable, which indicates that $|SC| \approx 0.13|V|$ in practice.

7.4 Iteration times

β	1.7	1.8	1.9	2.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7
|SC|/|V|	0.14	0.13	0.12	0.12	0.13	0.13	0.13	0.13	0.13	0.13	0.13

Figure 10: SC size with varied β

Data Set	One-k swap	Two-k swap
Astroph	6	3
DBLP	2	2
Youtube	4	4
Patent	7	6
Blog	5	8
Citeseerx	9	3
Uniport	9	4
Facebook	3	2
Twitter	6	4
Clueweb12	6	8

Table 7: Number of rounds in two algorithms

The third set of experiments was conducted to study the effect of each iteration of scan in ONE-K-SWAP and TWO-K-SWAP algorithms. Table 7 shows the total number of rounds, which is indicated by the running number of WHILE-LOOPs in Line 5 for each algorithm. Each round needs three iterations of sequential scans. We have the following observations: (1) The number of rounds varies from 2 to 9 for various data sets. But this number is not directly proportional to the size of graphs. For example, Facebook is much bigger than Astroph, but the number of rounds of Facebook is only half of that of Astroph (3 versus 6). (2) A surprising finding is that TWO-K-SWAP often takes less rounds than ONE-K-SWAP. Intuitively, TWO-K-SWAP handles more swap cases and is supposed to run more rounds than ONE-K-SWAP. But the empirical results in Table 7 is somewhat surprising. This can be explained that TWO-K-SWAP also includes the operation of one-k swaps, therefore TWO-K-SWAP can perform more swaps than ONE-K-SWAP in one round and thus stops earlier than ONE-K-SWAP. For example, for Twitter data, ONE-K-SWAP needs 6 rounds (while adding 621,183 new vertices into the independent set), but TWO-K-SWAP only needs 4 rounds to finish all swaps (adding 621,400 new vertices).

Early Stop: Table 8 shows the number of swapped vertices in first three rounds for various data sets in the ONE-K-SWAP algorithm. Due to the space constraint, we only report the results for ONE-K-SWAP whenever the results on TWO-K-SWAP exhibit similar trend. An interesting finding is that for all real data sets, more than 97% swaps have been finished within three rounds. This experiment

	Time					Memory cost				
Data Set	**DU**	**STXXL**	**Greedy**	**One-k**	**Two-k**	**DU**	**STXXL**	**Greedy**	**One-k**	**Two-k**
Astroph	129ms	73.6ms	57ms	347ms	237ms	4.43MB	25KB	4.5KB	149.1KB	329.7KB
DBLP	0.75s	1.40s	0.56s	1.36s	1.39s	128.3MB	0.25MB	51.9KB	1.65MB	3.55MB
Youtube	1.93s	2.67s	1.15s	3.78s	4.76s	239.1MB	1MB	141.6KB	4.59MB	9.69MB
Patent	21.3s	22.0s	4.6s	27.8s	36.7s	692.2MB	2MB	460.2KB	14.9MB	31.7MB
Blog	28.8s	30.0s	6.2s	35.7s	45.3s	841.9MB	2MB	493.2KB	15.9MB	34.4MB
Citeseerx	22.0s	16.0s	6.4s	25.7s	20.8s	1258.4MB	2MB	798.3KB	25.7MB	52.4MB
Uniport	18.6s	20.9s	2.2s	19.9s	18.5s	1242.7MB	2MB	850.8KB	27.5MB	55.4MB
Facebook	N/A	187.2s	47.9s	153.0s	160.8s	N/A	25MB	7.06MB	234.2MB	468.9MB
Twitter	N/A	18min	8min	39min	55min	N/A	25MB	7.34MB	242.2MB	524.1MB
Clueweb12	N/A	1.95h	1.65h	8.8h	10.4h	N/A	200MB	116.6MB	3.75GB	5.73GB

Table 6: Efficiency of various algorithms for real data sets (DU denotes DynamicUpdate)

demonstrates that the swap algorithm can stop earlier without much compromise for the results.

7.5 Evaluation on estimated bounds

Finally, we made experiments to evaluate the accuracy of the estimation (i.e. by Proposition 2 and 5) for the performance of our algorithms. Each algorithm is repeated 10 times and the average size of independent set is reported here. Table 9 shows the estimation results by the GREEDY algorithm (by Proposition 2), the real size of the independent set and the accuracy (= *Estimation*/*Real*). We have the following observations. First, our theoretical estimation is very tight. For all data sets, the accuracy is more than 98.7%. Second, consistent with our proof, this is a lower bound. Finally, very surprisingly, we find that the bigger the β, the smaller the size of independent set in GREEDY. This is somehow unexpected since the larger β means less edges in the graph and thus more vertices are supposed to be added into the independent set. (Note the total number of vertices in the graph is fixed.) But the empirical results are the contrary. After the deep thinking, we find that bigger β, more vertices with degree one are added into IS, but less vertices with degree > 1. The overall result is that the reduction surpasses the increase. Therefore, the total size of independent sets decreases.

β	**Edges**	**Estimation**	**Real**	**Accuracy**
1.7	215M	8,102,389	8,147,721	99.4%
1.8	118M	7,896,164	7,953,889	99.3%
1.9	72M	7,650,663	7,721,332	99.1%
2.0	49M	7,394,070	7,474,477	98.9%
2.1	36M	7,147,342	7,235,191	98.8%
2.2	29M	6,922,329	7,012,683	98.7%
2.3	24M	6,723,585	6,813,139	98.7%
2.4	21M	6,550,682	6,635,854	98.7%
2.5	18M	6,400,913	6,478,349	98.8%
2.6	17M	6,270,900	6,341,388	98.9%
2.7	15M	6,157,404	6,220,084	99.0%

Table 9: Accuracy of estimation for Greedy varying β

Summary From the comprehensive performance evaluation we have the following: (1) ONE-K-SWAP and TWO-K-SWAP algorithms, coupled with the ability of the optimization to swap vertices, lead to a large independent set (with high performance ratios: 96%-99.9%) using the limited main memory for massive graphs; (2) The performance advantage of swap operations is more pronounced based on the results of the Baseline or the STXXL approach; (3) The swap algorithms can stop earlier within three rounds of swaps (while completing 97%-100% swaps) to achieve a good balance in the tradeoff between the efficiency and the effectiveness of algorithms.

8. CONCLUSIONS AND FUTURE WORK

This work has studied the problem of the maximum independent set on massive graphs. Our solution starts with studying a delightfully simple yet effective greedy approach. We then propose two swap-based solutions, called ONE-K-SWAP and TWO-K-SWAP algorithms. We demonstrate that our algorithm can compute an independent set which is closely to the theoretical optimal bound for massive real-life graphs using very limited main memory. To the best of our knowledge, it is the first to provide a set of efficient solutions for MIS with the semi-external setting. There are several open directions for future work. We plan to investigate how our solutions can be extended to the incremental massive graphs with frequent updates. In long term, our efforts will aim at a proposal to study other graph problems like minimum vertex covers and graph coloring for massive graphs with a single commodity PC.

Acknowledgement: This research is partially supported by 973 Program of China (2012CB316205), NSF China (61472427) and the research fund from the University of Helsinki, Finland.

9. REFERENCES

[1] An implementation of the C++ standard template library STL for external memory computations. http://stxxl.sourceforge.net.

[2] J. Abello, A. L. Buchsbaum, and J. Westbrook. A functional approach to external graph algorithms. *Algorithmica*, 32(3):437–458, 2002.

[3] W. Aiello, F. R. K. Chung, and L. Lu. A random graph model for massive graphs. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 171–180, 2000.

[4] D. V. Andrade, M. G. C. Resende, and R. F. F. Werneck. Fast local search for the maximum independent set problem. *J. Heuristics*, 18(4):525–547, 2012.

[5] P. Berman and M. Fürer. Approximating maximum independent set in bounded degree graphs. In *SODA*, pages 365–371, 1994.

[6] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 595–601, Manhattan, USA, 2004. ACM Press.

[7] G. S. Brodal and R. Fagerberg. On the limits of cache-obliviousness. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 307–315, 2003.

[8] M. Brunato and R. Battiti. R-EVO: A reactive evolutionary algorithm for the maximum clique problem. *IEEE Trans. Evolutionary Computation*, 15(6):770–782, 2011.

[9] S. Butenko, P. M. Pardalos, I. Sergienko, V. Shylo, and P. Stetsyuk. Finding maximum independent sets in graphs arising from coding

Data Set	One round	Swap ratio	Two rounds	Swap ratio	Three rounds	Swap ratio	1-k swap	2-k swap
Astroph	716	72.69%	909	92.28%	960	97.46%	297ms	437ms
Dblp	131	100%	131	100%	131	100%	1.3s	2.0s
Youtube	2,466	98.13%	2,511	99.92%	2,513	100%	3.7s	6.1s
Patent	37,415	90.63%	40,756	98.72%	41,177	99.74%	16.9s	36.6s
Blog	20,629	96.90%	21,253	99.84%	21,286	99.99%	32.3s	49.8s
Citeseerx	16,096	85.51%	18,085	96.08%	18,597	98.80%	17.3s	29.8s
Uniport	3,941	86.54%	4,392	96.44%	4,505	98.92%	16.2s	26.3s
Facebook	4,906	99.98%	4,907	100%	4,907	100%	112s	208s
Twitter	1,164,794	90.10%	1,254,881	97.07%	1,276,892	98.78%	20min	46min
Clueweb12	106,682,409	91.02%	112,929,577	96.35%	115,601,912	98.63%	8.8h	10.4h

Table 8: Numbers of new IS vertices, swap ratios and the running time.

theory. In *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC), March 10-14, 2002, Madrid, Spain*, pages 542–546, 2002.

[10] U. Feige. Approximating maximum clique by removing subgraphs. *SIAM J. Discrete Math.*, 18(2):219–225, 2004.

[11] A. W.-C. Fu, H. Wu, J. Cheng, and R. C.-W. Wong. Is-label: an independent-set based labeling scheme for point-to-point distance querying. *PVLDB*, 6(6):457–468, 2013.

[12] X. Geng, J. Xu, J. Xiao, and L. Pan. A simple simulated annealing algorithm for the maximum clique problem. *Inf. Sci.*, 177(22):5064–5071, 2007.

[13] M. K. Goldberg, D. Hollinger, and M. Magdon-Ismail. Experimental evaluation of the greedy and random algorithms for finding independent sets in random graphs. In *Experimental and Efficient Algorithms, 4th InternationalWorkshop*, pages 513–523, 2005.

[14] M. M. Halldórsson and J. Radhakrishnan. Greed is good: approximating independent sets in sparse and bounded-degree graphs. In *STOC*, pages 439–448, 1994.

[15] P. Hansen, N. Mladenovic, and D. Urosevic. Variable neighborhood search for the maximum clique. *Discrete Applied Mathematics*, 145(1):117–125, 2004.

[16] J. Håstad. Clique is hard to approximate within n$^{1\text{-epsilon}}$. In *FOCS*, pages 627–636, 1996.

[17] M. Jiang, A. W. Fu, R. C. Wong, and Y. Xu. Hop doubling label indexing for point-to-point distance querying on scale-free networks. *PVLDB*, 7(12):1203–1214, 2014.

[18] S. Khanna, R. Motwani, M. Sudan, and U. V. Vazirani. On syntactic versus computational views of approximability. *SIAM J. Comput.*, 28(1):164–191, 1998.

[19] G. L. Nemhauser and W. T. Trotter. Vertex packing structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.

[20] J. M. Robson. Algorithms for maximum independent sets. *J. Algorithms*, 7(3):425–440, 1986.

[21] Y. Shen, D. T. Nguyen, and M. T. Thai. On the hardness and inapproximability of optimization problems on power law graphs. In *Combinatorial Optimization and Applications - 4th International Conference*, pages 197–211, 2010.

[22] T. Strijk, B. Verweij, and K. Aardal. Algorithms for maximum independent set applied to map labelling. Technical report, Universiteit Utrecht, 2000.

[23] J. S. Vitter. Algorithms and data structures for external memory. *Foundations and Trends in Theoretical Computer Science*, 2(4):305–474, 2006.

[24] P. Wan, X. Jia, G. Dai, H. Du, and O. Frieder. Fast and simple approximation algorithms for maximum weighted independent set of links. In *IEEE INFOCOM*, pages 1653–1661, 2014.

[25] V. K. Wei. A lower bound on the stability number of a simple graph. Technical report, Bell Labs Technical Journal, 1981.

[26] M. Xiao and H. Nagamochi. Exact algorithms for maximum independent set. In *Algorithms and Computation - 24th International Symposium*, pages 328–338, 2013.

[27] N. Zeh. I/O-efficient algorithms for shortest path related problems. Technical report, PhD thesis, Carleton University, 2002.

APPENDIX

1. Algorithm for the bound of the independence number.

Algorithm 5: Upper bound for the independence number

Input: A sorted adjacent-list file for graph G
Output: Upper bound B for the independence number of G

```
for v ∈ V of G do
    State[v] ← unVisited;
for v ∈ V of G do
    if State[v]=unVisited then
        N ← 0; State[v] ← visited;
        for u ∈ Ad(v) do
            if State[u]=unVisited then
                N++; State[u] ← visited;
        B = B+max{N,1};
Return B;
```

The purpose of the above algorithm is to calculate the upper bound of independent number of a graph. This algorithm is efficient, as it requires only one pass of the adjacent file.

2. Proofs for lemmas and theorems.

Proof of Lemma 1. (Sketch) The GREEDY algorithm prefers to select vertices with small degrees. Given any vertex v with degree i, if all i adjacent vertices of v have larger degrees than v, then v will be selected to the independent set in GREEDY. Assume that v denotes the xth vertex of degree i, Therefore, the possibility that v is added to the independent set is at least

$$\left(\frac{\left(\frac{e^\alpha}{i^{\beta-1}} - ix\right) + \sum_{s=i+1}^{\Delta} \frac{e^\alpha}{s^{\beta-1}}}{\sum_{s=1}^{\Delta} \frac{e^\alpha}{s^{\beta-1}}}\right)^i \tag{6}$$

where $\Delta = \lfloor e^{\frac{\alpha}{\beta}} \rfloor$ is the maximum degree of the graph. Summing up the possibilities of all $\lfloor \frac{e^\alpha}{i^\beta} \rfloor$ vertices with degree i immediately implies the expected number:

$$\begin{aligned} GR_i(\alpha,\beta) &\geq \sum_{x=1}^{\lfloor \frac{e^\alpha}{i^\beta} \rfloor} \left(\frac{\left(\frac{e^\alpha}{i^{\beta-1}} - ix\right) + \sum_{s=i+1}^{\Delta} \frac{e^\alpha}{s^{\beta-1}}}{\sum_{s=1}^{\Delta} \frac{e^\alpha}{s^{\beta-1}}}\right)^i \\ &= \sum_{x=1}^{\lfloor \frac{e^\alpha}{i^\beta} \rfloor} \left(\frac{\frac{ix}{e^\alpha} + \zeta(\beta-1,\Delta) - \zeta(\beta-1,i)}{\zeta(\beta-1,\Delta)}\right)^i \end{aligned} \tag{7}$$

which concludes the proof.

Proof of Lemma 3. It's sufficient to prove

$$\mathbf{Pr}(\exists v | v \in I_{1k} \wedge deg(v) \geq d_s) = o(\frac{1}{|V|}) \tag{8}$$

where I_{1k} denotes the new IS set after the 1-k swap algorithm. We first observe that, for any $d > 0$,

$$\begin{aligned}&\mathbf{Pr}(\exists v | v \in I_{1k} \wedge deg(v) = d) \\ &\leq \sum_{i=1}^{\lfloor \frac{e^\alpha}{d^\beta} \rfloor} \mathbf{Pr}(v_i \in I_{1k} \wedge deg(v_i) = d)\end{aligned} \tag{9}$$

Note that there are $\lfloor \frac{e^\alpha}{d^\beta} \rfloor$ vertices of degree d and let v_i be the ith vertex. If $v_i \in I_{1k}$, it has only one adjacent vertex in independent set before swap. Let $p = \mathbf{Pr}(|N(v_i) \cap I_{1k}| = 1 | deg(v_i) = d)$. Now we give an upper bound of p. Let $c(\alpha, \beta) = \frac{\sum_{i=1}^{\Delta} iGR_i(\alpha,\beta)}{e^\alpha}$. Further, it can be proved that $\sum_{v \in I_{1k}} deg(v) \geq \sum_{w \in I_{Greedy}} deg(w)$. Then

$$p \leq \frac{d \cdot c(\alpha, \beta)}{\zeta(\beta - 1, \Delta)} \left(\frac{\zeta(\beta - 1, \Delta) - 2c(\alpha, \beta)}{\zeta(\beta - 1, \Delta)} \right)^{d-1} \tag{10}$$

Let $c'(\alpha, \beta) = \frac{\zeta(\beta-1,\Delta)}{\zeta(\beta-1,\Delta)-2c(\alpha,\beta)}$. For $i \geq d$ we have

$$\frac{\mathbf{Pr}(\exists v | v \in I_{1k} | deg(v) = i + 1)}{\mathbf{Pr}(\exists v | v \in I_{1k} | deg(v) = i)} < \frac{1}{c'(\alpha, \beta)} \tag{11}$$

Note that

$$\begin{aligned}\mathbf{Pr}(\exists v | v \in I_{1k} | deg(v) \geq d) &\leq \sum_{i=d}^{\Delta} \mathbf{Pr}(\exists v | v \in I_{1k} | deg(v) = i) \\ &< \frac{c(\alpha, \beta)}{d^{\beta-1}\zeta(\beta - 1, \Delta)} \frac{c'(\alpha, \beta)}{c'(\alpha, \beta) - 1} \left(\frac{1}{c'(\alpha, \beta)} \right)^{d-1}\end{aligned} \tag{12}$$

Let the right side of Inequality (12) equal $\frac{1}{|V|}$, which implies the upper bound d_s as stated in Lemma 3.

Proof of Lemma 4. Prove by contradiction. Assume there exists $v' \in I$, $u \in ISN^{-1}(v')$ s.t. $deg(u) < deg(v')$. Since there is only one vertex in $N(u) \cap I$, i.e, v', according to the Greedy algorithm, u should be selected into I, but not v', a contradiction.

Proof of Proposition 5. (Sketch) Let $V_i = \{v | deg(v) = i\}$ and $A_i = \{u | deg(u) = i \wedge state(u) = A \wedge deg(ISN(u)) > 1\}$. Then $|V_i \cap I|$ can be estimated by $GR_i(\alpha, \beta)$, and $|A_i|$ can be estimated by $\sum_{v \in V_i} \mathbf{Pr}(v \in A_i)$. Note that $v \in A_i \Rightarrow v \in V_i - V_i \cap I$. Hence,

$$\begin{aligned}\mathbf{Pr}(v \in A_i | v \in (V_i - V_i \cap I)) &= \frac{Pr(v \in A_i)}{Pr(v \in (V_i - V_i \cap I))} \\ &\approx \frac{i \frac{c(\alpha,\beta)-1}{\zeta(\beta-1,\Delta)} \left(\frac{\zeta(\beta-1,\Delta)-2c(\alpha,\beta)}{\zeta(\beta-1,e^{\frac{\alpha}{\beta}})} \right)^{i-1}}{\sum_{j=1}^{i} \binom{i}{j} \left(\frac{c(\alpha,\beta)}{\zeta(\beta-1,\Delta)} \right)^{j} \left(\frac{\zeta(\beta-1,\Delta)-2c(\alpha,\beta)}{\zeta(\beta-1,\Delta)} \right)^{i-j}}\end{aligned} \tag{13}$$

Further, let $A_{i,j} = \{v | v \in A_i \wedge ISN(v) \in V_j \cap I\}$. Note that $j \leq i$ by Lemma 4. By evenly distributing these vertices we get $|A_{i,j}| \approx \frac{j|V_j \cap I|}{\sum_{x=2}^{i} x|V_x \cap I|}$. Suppose there are m_1 type-1 balls and m_2 type-2 balls. The number of bins is n and the size of each bin is limited to d. Then $\mathbf{Pr}(m_1, m_2, n, d)$ denotes the probability of an event that the first bin contains at least one type-1 ball and one type-2 ball. Then

$$\mathbf{Pr}(m_1, m_2, n, d) = \frac{\binom{d}{1}\binom{n-d}{m_1-1}\binom{d-1}{1}\binom{n-d-m_1+1}{m_2-1}}{\binom{n}{m_1}\binom{n-m_1}{m_2}} \tag{14}$$

Therefore, the number of new vertices is:

$$T(x, y, i) = |V_i \cap I| \mathbf{Pr}(|A_{x,i}|, |A_{y,i}|, |V_i \cap I|, i), x, y \geq i \tag{15}$$

Proof of Lemma 6. (Sketch) We first compute the maximal degree d_{2k} for vertices in SC. A non-IS vertex with large enough degree should have more than two IS neighbours with high probability. Let I denote the IS vertices after the Greedy algorithm. If a vertex u has more than two adjacent vertices in I, then state(u)$\neq$"A", and thus u cannot contribute to any SC set. Thus, we have

$$\begin{aligned}&\mathbf{Pr}(|N(u) \cap I| > 2 | u \notin I \wedge (deg(u) \geq d_{2k})) = 1 - o(\frac{1}{|V|}) \\ &= \frac{\mathbf{Pr}(|N(u) \cap I| > 2 \wedge (deg(u) \geq d_{2k}))}{\mathbf{Pr}(u \notin I \wedge deg(u) \geq d_{2k})} = \frac{\sum_{i=3}^{d_{2k}} p_i}{\sum_{i=1}^{d_{2k}} p_i}\end{aligned} \tag{16}$$

where $p_i \approx \binom{d_{2k}}{i} \left(\frac{c(\alpha,\beta)}{\zeta(\beta-1,\Delta)} \right)^{i} \left(\frac{\zeta(\beta-1,\Delta)-2c(\alpha,\beta)}{\zeta(\beta-1,\Delta)} \right)^{d_{2k}-i}$ denotes the probability that a non-IS vertex has i adjacent IS vertices. Note that $c(\alpha, \beta) = \frac{\sum_{i=1}^{\Delta} iGR_i(\alpha,\beta)}{e^\alpha}$ is defined in Lemma 3.

$$d_{2k} < \frac{\alpha + ln\zeta(\beta, \Delta) + 2ln\frac{\zeta(\beta-1,\Delta)}{\zeta(\beta-1,\Delta)-c(\alpha,\beta)}}{ln\frac{\zeta(\beta-1,\Delta)-c(\alpha,\beta)}{\zeta(\beta-1,\Delta)-2c(\alpha,\beta)}} \tag{17}$$

Given a vertex pair $(u, v) \in SC(w_1, w_2)$, there are two cases: 1) both $ISN(u)$ and $ISN(v)$ equal $\{w_1, w_2\}$ and; 2) only $ISN(u)$ equals $\{w_1, w_2\}$. For the first case, the total number of vertices in SC is bounded by the number of all vertices which have two adjacent IS vertices, i.e., $\sum_{i=2}^{d_{2k}} |V_i - I| \frac{p_2}{\sum_{j=1}^{i} p_i}$.

For the second case, given a vertex v, $ISN(v)$=$\{w_1\}$, if a vertex u can be added into SC set with v, then u should have two IS neighbours and one of them is w_1. Then the number of vertices of u is bounded by $\sum_{i=2}^{d_{2k}} ib_i \leq b_{max} \sum_{i=2}^{d_{2k}} i$, where

$$b_{max} = \frac{1}{log\frac{\zeta(\beta-1,\Delta)}{\zeta(\beta-1,\Delta)-2c(\alpha,\beta)}} \frac{c(\alpha, \beta)}{\zeta(\beta - 1, \Delta)} \tag{18}$$

So the total number of vertices in SC for the second case is bounded by $b_{max} \sum_{i=2}^{d_{2k}} i|V_i - I| \frac{p_1}{\sum_{j=1}^{i} p_i}$.

Since $|V_i - I| < |V_i|$, by summing up two cases, we get

$$|SC| < \sum_{i=2}^{d_{2k}} |V_i| \frac{ib_{max}p_1 + p_2}{\sum_{j=1}^{i} p_i} \tag{19}$$

Therefore, $|SC| < \sum_{i=2}^{d_{2k}} |V_i| < |V| - |V_1| = |V| - e^\alpha$, which concludes the proof.

Time Complexity of the TWO-K-SWAP Algorithm. As mentioned in the last paragraph of Section 6, we aim to prove that degree(w_i) = $O(log|V|)$, that is the degree of each vertex in SC is $O(log|V|)$.

Note that the proof of Lemma 6 shows that degree(w_i)$\leq$ d_{2k}. Therefore, we need to prove that $d_{2k} = O(log|V|)$. See Equation (17). Note that $\alpha + ln\zeta(\beta, \Delta) = ln|V|$. Since $c(\alpha, \beta) < \frac{1}{2}\zeta(\beta - 1, \Delta)$, $2ln\frac{\zeta(\beta-1,\Delta)}{\zeta(\beta-1,\Delta)-c(\alpha,\beta)} < 2ln(2) \approx 1.58$. Note that $c(\alpha, \beta) = \frac{\sum_{i=1}^{\Delta} iGR_i(\alpha,\beta)}{e^\alpha}$. It can be proved that $c(\alpha, \beta)$ is monotonically increasing in α and monotonically decreasing in β. For $\beta \leq 2.7$ and a sufficient large α, say 15, it's easy to see that $c(\alpha, \beta) > 0.8\zeta(\beta, \Delta)$ holds. Let $\bar{d} = \frac{\zeta(\beta-1,\Delta)}{\zeta(\beta,\Delta)}$. Hence, $ln\frac{\zeta(\beta-1,\Delta)-c(\alpha,\beta)}{\zeta(\beta-1,\Delta)-2c(\alpha,\beta)} > ln\frac{\bar{d}-0.8}{\bar{d}-1.6}$. Since $2 < \bar{d} < 20$ for $1.7 \leq \beta \leq 2.7$, $ln\frac{\bar{d}-0.8}{\bar{d}-1.6} \in (0.04, 1.1)$. Therefore, $d_{2k} = \frac{log(|V|)+(0,1.58)}{(0.04,1.1)} = O(log|V|)$, as desired.