Universiteit Leiden
The Netherlands

Opleiding Informatica

Comparing Two Algorithms for Finding

Maximal Matchings in Bipartite Graphs

Rachel de Jong

Supervisors:
Alfons Laarman & Lieuwe Vinkhuijzen

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

16/07/2019

Abstract

Two different algorithms for finding maximal matchings in bipartite graphs will be compared: the algorithm of Hopcroft and Karp and a new algorithm, Optimistic Hopcroft and Karp, which is based on the algorithm of Hopcroft and Karp. This new algorithm has the possibility to optimistically look further for augmenting paths than the algorithm of Hopcroft and Karp with as aim to find more augmenting paths in each iteration.

Contents

Chapter 1

Introduction

The maximal matching problem is a problem for which the aim is to find an as large as possible set of edges such that each node in the graph occurs in at most one edge in this set. There are two different approaches for finding maximal matchings in graphs: first, the algebraic approach [KUW86] and second, the approach that looks for augmenting paths, which is the approach that this bachelor thesis will look into.

One of the first algorithms using the second approach for finding maximal matchings was created by Edmonds [Edm65] for bipartite graphs. This algorithm also has been extended for general graphs. In 1973 Hopcroft and Karp created a more efficient algorithm (for bipartite graphs) with time complexity $\mathcal{O}(\sqrt{|N|}|E|)$ (with N the number of nodes and E the number of edges) [HK73] that finds augmenting paths more efficiently: instead of looking for one augmenting path at a time, this algorithm finds a maximal set of disjoint augmenting paths in each iteration. This algorithm has also been extended for general graphs in 1989 by Micali and Vazirani [MV80] with time complexity $\mathcal{O}(\sqrt{|N|}|E|)$ and in 1990 by Norbert Blum [Blu90] with time complexity $\mathcal{O}(\sqrt{|N|}|E|)$.

Lieuwe Vinkhuijzen has created a new algorithm, *"Optimistic Hopcroft and Karp"*, which is an adaptation of the algorithm of Hopcroft and Karp, to find maximal matchings in bipartite graphs. The aim of this thesis is to compare this new algorithm to the algorithm of Hopcroft and Karp to find out which algorithm is better. Therefore the research question for this bachelor thesis is: Comparing two algorithms for finding maximal matchings in random graphs, the algorithm of Hopcroft and Karp and Optimistic Hopcroft and Karp, which algorithm performs better?

1.1 Thesis Overview

In chapter 2 some background will be described containing definitions and a more elaborate description of the problem. Next, both the algorithm of Hopcroft and Karp and the new algorithm will be described in chapter 3. In chapter 4 the executed experiments will be described and in chapter 5 the results of these experiments will

be discussed. In chapter 6 some conclusions will be drawn.

Chapter 2

Background

2.1 Definitions

Definition 1 (Undirected Bipartite Graph)**.** An undirected bipartite graph G = (N, E) is a set of edges E and nodes N such that:

- The nodes in N can be partitioned in two sets A and B such that there are only edges from nodes in A to nodes in B and from nodes in B to nodes in A. (Bipartite)
- For all edges: if $(i, j) \in E$ then $(j, i) \in E$. (Undirected)

Definition 2 (Matching)**.** A matching is a set of edges such that every node in the graph occurs in at most one edge in the matching.

Definition 3 (Free Node)**.** A node is free, relative to a matching M, if it does not occur in an edge in M.

Definition 4 (Matched Node)**.** A node is matched, relative to a matching M, if it does occur in an edge in M.

Definition 5 (Maximal Matching)**.** A matching for a graph is maximal if no larger matching exists for that graph.

Definition 6 (Perfect Matching)**.** A matching is perfect if each node in the graph occurs in exactly one edge in the matching.

Definition 7 (Alternating Path)**.** An alternating path with respect to matching M is a path between two nodes such that the edges are alternatingly in the matching and not in the matching. Three different types of alternating paths can be distinguished, which are also illustrated in Figure 2.1:

1. Paths that start and end with a node that is not free
2. Paths that start with a node that is free and end with node that is not free or start with a node that is not free and end with a node that is free
3. Paths that start and end with a node that is free

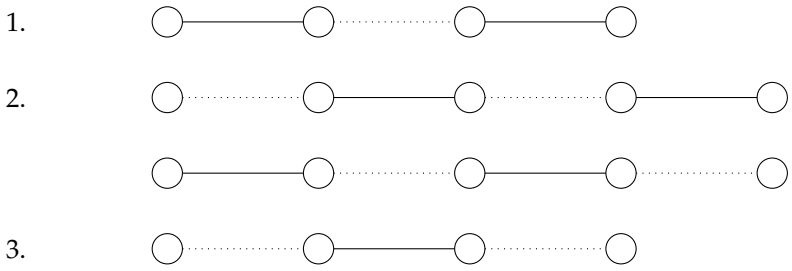

Figure 2.1: Three types of alternating paths

Definition 8 (Augmenting Path)**.** An alternating path that both starts and ends with a free node is called an augmenting path: this is the third type of alternating paths described.

Definition 9 (Diameter of a Graph)**.** The diameter of a graph G is the longest shortest path between two nodes in G. If there is no path between two nodes in a graph, the length of this path is undefined.

In the algorithms discussed, the breadth first search technique is used. This is a graph traversel such that each iteration all neighbours of the current nodes are visited and next all those neighbours etc. A Breadth First Search stops when all vertices that can be reached from the starting node have been visited. It is described in more detail in [CLRS09].

BFS also refers to the step of the algorithm that generates a directed graph, which uses this technique. More details on this step are described in subsection 3.1.1.

In the algorithms discussed, the depth first search technique is used as well. This is a graph traversel that starts at a node, visits some neighbour until it can not reach any other node and then backtracks to visit another node until all nodes that can be reached from the starting node have been visited. It is described in more detail in [CLRS09].

This is a technique used in the algorithms discussed and also refers to the step of the algorithm that finds a maximal set of disjoint paths in the given directed graph, which uses this technique. More details on this step are described in subsection 3.1.2.

2.2 Edmond's Algorithm

Theorem 1 (Berge's Theorem [Ber57])**.** M is a maximal matching for a graph if and only if there is no augmenting path with respect to M.

Both of the algorithms that will be compared find a maximal matching by finding augmenting paths. If an augmenting path P with respect to matching M has been found, $M \oplus P$ can be determined (as illustrated in Figure 2.2) which always results in a matching with one more edge.

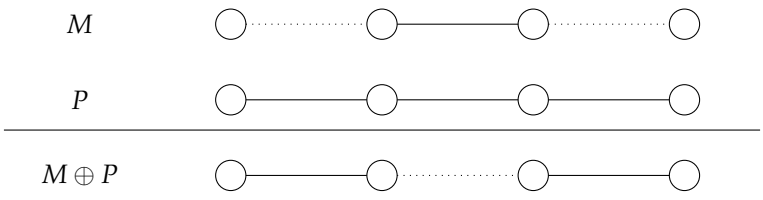

Figure 2.2: Example of determining $M \oplus P$

Combining that with Berge's theorem, a simple algorithm can be created for finding a maximal matching in a graph: Edmond's algorithm [Gre06], which can be seen in Algorithm 1. The algorithm keeps on looking for an augmenting path and if one is found, the symmetric difference between the path and the matching is computed. When no path can be found, the algorithm is finished and a maximal matching has been found.

Algorithm 1: Edmond's Algorithm

Input: Graph G
Result: Matching M containing a maximal matching
$M = \emptyset$
while *There is an augmenting path with respect to* M **do**
    P = Augmenting path with respect to M
    $M = M \oplus P$
end
return M

2.3 Graphs

For comparing the algorithms, undirected bipartite random graphs will be used.

Definition 10 (Undirected Random Bipartite Graph)**.** A random undirected bipartite graph is a graph containing nodes with which there is a chance of p% with $p \in [0, 100]$ that there exists an edge between two nodes i and j where $i \in A$ and $j \in B$. This graph is also undirected and is generated as described in Algorithm 2.

Algorithm 2: Generation of an Undirected Random Bipartite Graph

Input: Probability p, number of nodes n
Result: An Undirected Random Bipartite Graph $G = (N, E)$
$N = 0, 1 \ldots n - 1$
for $i = 0$ *to* $(n/2) - 1$ **do**
    for $j = n/2$ *to* $n - 1$ **do**
        if $r \in [0, 100] \leq p$ **then**
            $E = E \cup \{(i, j)\}$
            $E = E \cup \{(j, i)\}$
        end
    end
end
return G

It is possible to generate random graphs with different values of p which will result in graphs with different numbers of edges and different structures. Erdős and Rényi wrote about the evolution of general random graphs when the number of edges that is contained is increased [ER60]. They distinguished five different phases:

1. In the first phase the graph mainly consists of components that are trees.
2. During the second phase, the graph does contain cycles. It mainly consists of trees and components where the number of nodes equals the number of edges in the component.
3. In the third phase the structure of the random graph changes abruptly. In most cases there is a giant

component with $N^{2/3}$ nodes with a very complex structure. The other components are relatively small components, of which most are trees. During this phase the small components melt into the giant component, with the smaller components having a larger chance of survival.

4. During the fourth phase the graph almost surely becomes connected. It is also almost certain that all components outside the giant component are trees of order $\leq$ k.

5. During the last phase the graph is almost surely connected.

Erdős and Rényi also found that there is a threshold for perfect matchings in [ER66]. For bipartite graphs in the book [FK15], which is a good survey about random graphs, the following theorem has been given:
Theorem 2 (Perfect Matchings in Bipartite Graphs [FK15])**.** Let $\omega = \omega(N)$, some function with N as input, $c > 0$ be a constant, and $p = \frac{^e log(N)+\omega}{N}$. Then

$$\lim_{N\to\infty} \mathbb{P}(\mathbb{G}_{A,B,p} \; has \; a \; perfect \; matching) \begin{cases} 0 & if \; \omega \to -\infty \\ e^{-2e^{-c}} & if \; \omega \to c \\ 1 & if \; \omega \to \infty \end{cases}$$

Furthermore, in [ER66] Erdős and Rényi described that, for general random graphs, if the number of edges $E = 1/2 \times N \times^e log(N) + O(N)$, then the graph G with E edges and N nodes has a probability near 1 to consists of a number of connected nodes and a number of isolated nodes.

Chapter 3

Algorithms

In this chapter the two algorithms that will be compared are described in detail. The first algorithm is the algorithm of Hopcroft and Karp (H&K) and is described in section 3.1. The second algorithm, Optimistic Hopcroft and Karp (OH&K), is discussed in section 3.2.

3.1 Hopcroft and Karp

The algorithm of Hopcroft and Karp (H&K) was created in 1973 and has a time complexity of $\mathcal{O}(n^{5/2})$ [HK73]. It is still the most efficient algorithm for finding maximal matchings in bipartite graphs. The idea of this algorithm is that it will find augmenting paths more efficiently than Edmond's algorithm, which is described in section 2.2. Instead of finding only one augmenting path during each iteration, this algorithm will find a maximal set of (node-)disjoint paths of a certain length during each iteration. This set is maximal such that no more augmenting paths of a certain length can be added to the set such that it is disjoint from the other paths; it is not necessarily true that it finds the largest set of augmenting paths given a graph and the current matching. The algorithm consists of three parts as described in Algorithm 3. In the upcomming subsections each step of this algorithm will be described in detail.

1. Build a directed graph.
2. Find a maximal set of (node-)disjoint augmenting paths in this directed graph.
3. Compute $M = M \oplus P$, which will result in a larger matching (if $P \neq \emptyset$).

The first two steps can be seen as the steps for finding augmenting paths and the last step is computing the symmetric difference. In that way, the structure of H&K and Edmond's algorithm are quite similar.

Algorithm 3: Hopcroft and Karp

Input: graph $G = (N, , E)$
Result: Matching M containing a maximal matching
$M = \emptyset$;
while *There is an augmenting path with respect to* M **do**
    H = BuildDirectedGraphWithBFS(G, M)
    P = AlternatingDepthFirstSearch(H, M)
    $M = M \oplus P$
end
return M;

3.1.1 Directed Graph

The first step of H&K is generating a directed graph. The goal of generating this graph is that in the next step it can be used to find a maximal set of disjoint augmenting paths of minimal length for the graph and current matching. The creation of this directed graph may also be referred to as BFS since it uses the Breadth First Search technique.
The required inputs for executing this step are:

- An undirected bipartite graph G with a set of nodes N and a set of edges E.
- A matching M.
- A partition of nodes in G in sets A and B such that all edges in E are from A to B or from B to A.

Pseudocode for generating this directed graph can be seen in Algorithm 4 and works as follows:

1. The algorithm starts with the directed graph G' with the nodes N' and edges E' containing no elements. First the set L_0, with L_i denoting the set of nodes on level i, is determined which contains all free nodes in A. Those nodes are added to N'. Then from those nodes a Breadth First Search is started with $i = 0$.

2. (a) The Breadth First Search is continued from all nodes a in L_i (with i is even) and for all edges $(a, b) \in E \wedge (a, b) \notin M \wedge (b \notin N' \vee b \in L_{i+1})$ the edge (b, a) is added to E' and if $b \notin N'$, node b is added to N' and to the set L_{i+1}. Then step 2b is executed.

   (b) If there is a node $b \in L_{i+1}$ such that b is free, the algorithm will halt after the Breadth First Search on this level is completed. Then step 3 is executed with $i = i + 1$.

3. The Breadth First Search is continued from all nodes b in L_i (with i is odd) and for all edges $(a, b) \in E \wedge (a, b) \in M \wedge (a \notin N' \vee a \in L_{i+1})$ the edge (a, b) is added to E' and if $a \notin N'$, node a is added to N' and to the set L_{i+1}. Then go back to step 2a with value $i = i + 1$.

This algorithm stops either when a directed graph has the depth of the shortest augmenting path or if no more nodes can be visited.

Algorithm 4: Pseudocode BuildDirectedGraphWithBFS

Input: graph $G = (A \cup B, E)$, matching M, the set of free nodes in A A'
Result: Directed graph G' containing augmenting paths
int $currentLength = 0$
bool $minLength = false$
set $V, Vnew, Visited = \emptyset$
$G' = (A', \emptyset)$
$V = A'$
for *all nodes in* A'*:* i **do**
  add node i to G'
end
while $V \neq \emptyset$ & $minLength == false$ **do**
  for *all nodes in* V*:* i **do**
    if $i \notin Visited$ **then**
      for *all edges* (i, j) **do**
        if $j \notin Visited$&$((currentLength\%2 == 0\&(i,j) \notin M)$
        or $(currentLength\%2 == 1\&(i,j) \in M)))$ **then**
          if $j \notin N'$ **then**
            $N' = N' \cup \{j\}$
          end
          $E' = E' \cup \{(j, i)\}$
          $Vnew = Vnew \cup \{j\}$
          if j *is a free node &* $currentLength$ %2 $== 0$ **then**
            $minLength = true$ //Augmenting path has been found
          end
        end
      end
    end
    $Visited = Visited \cup \{i\}$
  end
  $currentLength++$
  $V = Vnew$
  $Vnew = \emptyset$
end
return G'

The generated directed graph G' with nodes N' and edges E' has the following properties after it is completed:

- G' is acyclic.
- Every node in G corresponds to at most one node in directed graph G'.
- For all edges in E': $E' = \{(b, a) | (a, b) \in E - M, a \in A, b \in B\} \cup \{(a, b) | (a, b) \in M, a \in A, b \in B\}$ Thus:
  - G' contains only alternating paths.
  - All nodes on the odd level are in B (starting from the free nodes in L_0 at level 0).
  - All nodes on the even levels are in A (starting from the free nodes in L_0 at level 0).
- The length of the longest path in G' is the same as the length of the shortest augmenting path in G. This length will be referred to as i^* with $i^* = min\{i | L_i \cap free\ nodes\ in\ B \neq \emptyset\}$.

- All free nodes in L_{i^*} are the start of at least one augmenting path.

3.1.2 Finding a Maximal Set of Disjoint Paths

The second step is to find a maximal set of disjoint paths of a certain legnth, for which pseudocode can be found in Algorithm 5. This set is, again, maximal in the way that no more augmenting path can be added to the set such that the paths in the set are disjoint. This is done by starting depth first searches from all nodes in L_{i^*}. It is implemented as follows:

1. Start with two sets t = free nodes in L_0, s = free nodes L_{i^*}.
2. While s and t are not empty: choose a node b from s. This could be implemented as a random choice. However in the implemented code it is not random which makes it easier to analyze the algorithms.
3. Using DFS, try to find a path from $b \in s'$ to a node $a \in t'$.
4. If such a path has been found, add it to the set of paths p and delete all nodes and edges in p from G'.
5. Repeat from step 2.

3.2 Optimistic Hopcroft and Karp

In this section the new algorithm for finding maximal matchings will be described. This algorithm is similar to H&K. The main difference is that this algorithm does not always stop building the directed graph when an augmenting path of minimal length is contained in the graph; instead, it calculates a value ℓ. If this value ℓ is larger than or equal to the length of the shortest augmenting path, the algorithm stops when it reaches depth ℓ. If ℓ is smaller than the minimal length of an augmenting path, it stops, like H&K, at the depth of the shortest augmenting path. The idea is that this algorithm may optimistically look further than H&K and therefore might find a larger set of disjoint augmenting paths during each iteration. Hence, this new algorithm may require less iterations and could be more efficient than H&K. An overview of the algorithm can be seen in Algorithm 6 below.

The first step of this algorithm is to choose a clever value for ℓ. Which raises the question: "What would be a good value for ℓ?". In chapter 4 the experiments in which some possible values for ℓ are tested are described. To calculate the value of ℓ, the following formula will be used:

$$\ell = r \times (2\frac{k}{m-k}) + 1$$

Algorithm 5: Alternating Depth First Search

Input: Undirected graph $G' = (N', E')$
Result: A maximal set of disjoint paths in G': P
set $p, P, Visited = \emptyset$
S = free nodes in L_{i^*}
T = free nodes in L_0
while $S \neq \emptyset$ & $T \neq \emptyset$ **do**
  i = a node in S
  $S = S/i$
  $p = p \cup i$
  $Visited = Visited \cup i$
  while $p \neq \emptyset$ **do**
    if $i \in T$ **then**
      $T = T/i$
      $Visited = Visited \cup i$
      $p = p \cup i$
      $P = P \cup p$
      while $p \neq \emptyset$ **do**
        $N = N' - p.top$
        $p = p/p.top$
      end
    end
    else
      if *i has at least one neighbour* **then**
        i = an unvisited neighbour of i
        $p = p \cup i$
        $Visited = Visited \cup i$
      end
      else
        $p.pop$
      end
    end
  end
end
return P

With:

- ℓ = Number of edges in the path.
- r = A parameter.
- m = Size of the maximal matching.
- k = Size of the current matching.

Theorem 3. $\frac{k}{m-k}$ is a lower bound for the average number of matched edges in an augmenting path in the set of disjoint augmenting paths found for the current graph for the current matching if all matched edges are used for finding a new set of augmenting paths.

Proof 1. We know that 1) All k matched edges are used for finding a new set of augmenting paths. 2) If $m - k$ matchings are found, the matching is maximal; therefore at most $m - k$ augmenting paths can be found.
As a result k edges are divided over at most $m - k$ paths. Thus under the assumption that all matched edges are used, $\frac{k}{m-k}$ denotes a lower bound for the average number of matched edges in an augmenting path in the set of disjoint augmenting paths found for the current graph for the current matching.

Algorithm 6: Optimistic Hopcroft and Karp

Input: Graph $G = (N, E)$
Result: Matching M containing a maximal matching
$M = \emptyset$ **while** *There is an augmenting path with respect to* M **do**
    ℓ = CleverValue(G, M)
    H = BuildDirectedGraphWithBFSL(G, M, ℓ)
    P = AlternatingDepthFirstSearch(H, M)
    $M = M \oplus P$
end
return M

The total formula then denotes $r\times$ the lower bound of the average augmenting path length: counting both matched and unmatched edges. Since augmenting paths always have an odd number of edges, the value of ℓ is always odd.

The second step of the algorithm is to generate a directed graph; this is the same as for H&K except that it continues until the chosen depth ℓ has been reached instead of at the depth of the shortest length of the augmenting paths in the graph. The pseudocode can be found in Algorithm 4 where the condition in the first while loop should be replaced by ”v != $\emptyset$ & !(minLength & currentLength $\geq \ell$)”. The remaining two steps, finding a set of disjoint augmenting paths and computing the symmetric difference, are the same as for H&K and are described in section 3.1.

Chapter 4

Methods

In this chapter the performed experiments are described. The main goal of these experiments is to find out which algorithm is better. To do this, both algorithms have been implemented in C++ using the Snap library [Les99]. A benchmark, with which various experiments can be executed on both algorithms and data can be generated, has been implemented as well and has been used to conduct the experiments.

For the experiments bipartite random graphs are used with 1000 nodes and various values of p. The random graphs are generated according to Algorithm 2 in section 2.3. As the results will show, more difficult random graphs to find perfect matchings for are found in the range of $p = 0$ to $p = 2$. All five phases distinguished by Erdős and Rényi [ER60] occur in this range; the fifth phase starts at $p = 1$ for a graph with 1000 nodes, which makes the graphs in this range even more interesting. Therefore results for random graphs with smaller values of p will be discussed as well as values of p in the range of 0 to 100.

The conducted experiments can be grouped in two different categories with each a different purpose: The first category aims to learn more about random graphs, the second one to learn more about the behaviour of both algorithms and to compare them.

1. Random graphs
   (a) Perfect matching threshold.
   (b) Size of largest component.
   (c) Diameter of graphs.
2. Algorithms
   (a) How many paths are found during which iteration.
   (b) Find a good value for ℓ by testing different values for r.
   (c) Compare the algorithms on random graphs.

4.1 Perfect Matching Threshold

For this experiment a number of random graphs will be tested for each value of p in the range of 0 to 3 and for each value 100 random graphs will be generated for which the algorithm of Hopcroft and Karp will try to find a perfect matching. The results will show in how many cases the graphs contain a perfect matching and the size of the found matching. The size of the found matchings in the range of 0 to 2 will be shown as well. The found results will be compared with the theory of Erdős and Rényi discussed in section 2.3.

4.2 Size of Largest Component

Erdős and Rényi [ER60] used the term "giant component" to describe the evolution of random graphs. For this experiment the size of the largest component in the random graph will be measured for different values of p in a range of $p = 0$ to $p = 2.5$.

4.3 Diameter of Graphs

For this experiment we will look at the diameter of random graphs with different values for p. To calculate this, the algorithm of Frank Takes [TK11] has been used.

4.4 Augmenting Paths Per Iteration

To get a better understanding of the behaviour of the algorithms, a small experiment has been conducted. For this experiment both the algorithms of Hopcroft and Karp (H&K) and Optimistic Hopcroft and Karp (OH&K) with $r = 5$ are used. The aim of this experiment is to find how many augmenting paths are found during which iteration. This is done by running both algorithms on 30 different random graphs for both values $p = 0.5$ and $p = 50$ and calculating the average number of augmenting paths found during each iteration.

4.5 Comparing the Algorithms

For this experiment we will look at the behaviour of H&K and OH&K, using values of ℓ with $r = 0$ and $r = 5$, and the algorithms will be compared with each other. Results will be shown for both a range of p from $p = 0$ to $p = 100$ with a stepsize of 1 and $p = 0$ to $p = 2$ with a stepsize of 0.1. For all values of p, a number of random graphs will be generated on which all algorithms will run. Various values are measured during execution of the algorithms of which the average of each measurement will be reported in the results:

- The execution time.

- The number of iterations required.
- The number of edges visited in the BFS step.
- The size of directed graphs generated.
- The number of edges visited in the DFS step.
- The length of all augmenting paths found.

For all of these results the average will be shown and the average $\pm$ 3 times the standard deviation.

4.6 Value for ℓ

The new algorithm contains a formula to compute a value for ℓ with parameter r:

$$\ell = r \times (2\frac{k}{m-k}) + 1$$

With:
- ℓ = Number of edges in the path.
- m = Size of the maximal matching.
- k = Size of the current matching.
- r = An unknown parameter.

For the experiment the following values of r will be compared:

- 0 to 5: 0, 1, 2, 3, 4, 5
- 0 to 1: 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0

The larger the value of r, the further the algorithm can look for augmenting paths. OH&K with $r = 0$ will, like the algorithm of Hopcroft and Karp, only find augmenting paths of minimal length. Furthermore, since the size of the maximal matching is also unknown before and during execution of the algorithm, it is assumed that the size of the maximal matching equals the size of a perfect matching: thus m equals the number of nodes in the graph divided by two.

Both ranges of possible values for r will be tested on the range $p = 0$ to $p = 2$ with stepsize 0.1. For each value of p, 30 random graphs will be generated for which all algorithms will find a maximal matching. During execution, various values are measured of which the average of each measurement will be reported in the results:

- The execution time.
- The number of iterations required.
- The number of edges visited in the BFS step.

Chapter 5

Results

5.1 Perfect Matching Threshold

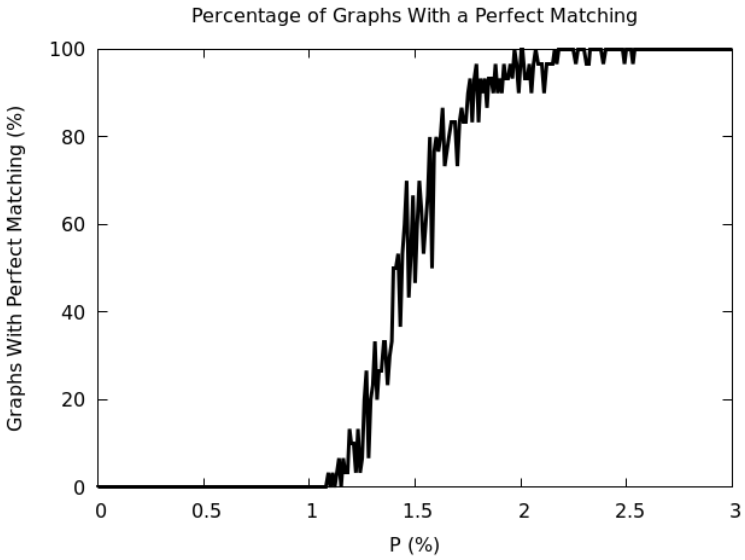

Figure 5.1: Perfect matching threshold

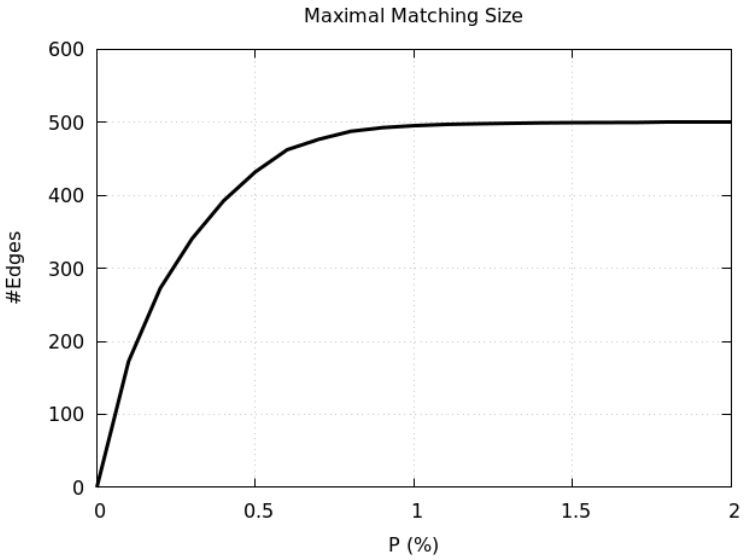

Figure 5.2: Number of augmenting paths found

According to the theory of Erdős and Rényi as discussed in section 2.3, if the probability p that an edge is contained in the graph equals $p = \frac{^e log(N)+\omega}{N}$ with $omega \to c$ with $c > 0$, the random graph may contain a perfect matching. For 1000 nodes, this equals $p > \frac{^e log(1000)}{1000} \approx 0.007+ \approx 0.7\%$.

This does not completely corresponds to the results illustrated in Figure 5.1. While with $p < 1.1$ there are no random graphs found that contain a perfect matching, after $p = 1.1$ there is a very small chance that the random graph generated contains a perfect matching and this chance increases as the value for p grows further. Around $p = 2.2$ the random graphs are almost certain to contain a perfect matching. In Figure 5.2 it can be seen that before the random graphs contain a perfect matching, the size of the maximal matching is already quite large and close to the size of a perfect matching, which is also the case for $p = 0.79\%$. Around $p = 1$,

which is where the random graphs start to have a small chance to contain a perfect matching, the matchings found are almost perfect.

5.2 Size of the Largest Component

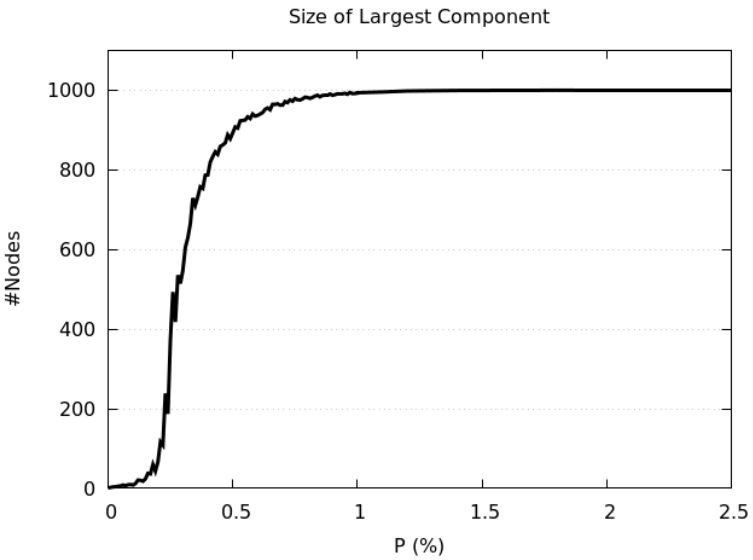

Figure 5.3: Size of largest component in random graphs

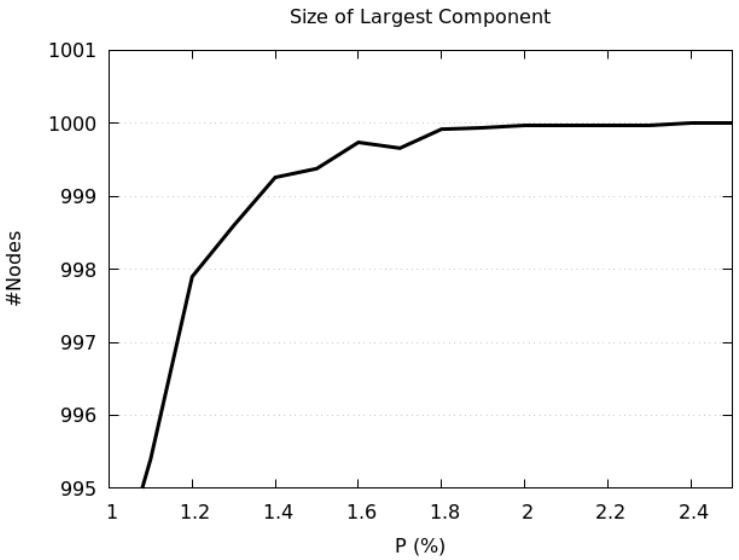

Figure 5.4: Size of largest component in random graphs

According to the theory of Erdős and Rényi as discussed in section 2.3, if the number of edges in the graph is $E = 1/2 \times N \times^e log(N) + O(N)$, the random graphs is very likely to consist of a single large component and various isolated nodes.

For 1000 nodes, this corresponds to $N = 1000$ thus $E = 1/2 \times 1000 \times^e log(1000) \approx 3454$ and thus $p \approx 1.38$. In Figure 5.3 and 5.4 it can be seen that in the case of bipartite random graphs the largest component contains almost all nodes at $p = 1.0$. At the value of $p = 1.38$, the average largest component contains all nodes, except one, which is isolated.

5.3 Diameter of Graphs

According to Figure 5.5, which shows results for the diameter of 5 different random graphs for each value for p, random graphs with a value of $p > 20$ are very likely to have a diameter of 3, except for $p = 100$ with a diameter of 2. On the other hand, as illustrated by Figure 5.6 where the average diameter of 50 different random graphs is plotted with $\pm$ 3 times the standard deviation, random graphs with a smaller value of p, especially in the range $p[0, 1\%]$, overall tend to have a larger diameter with a maximum of 20 at $p = 0.4$ that decreases after that value of p.

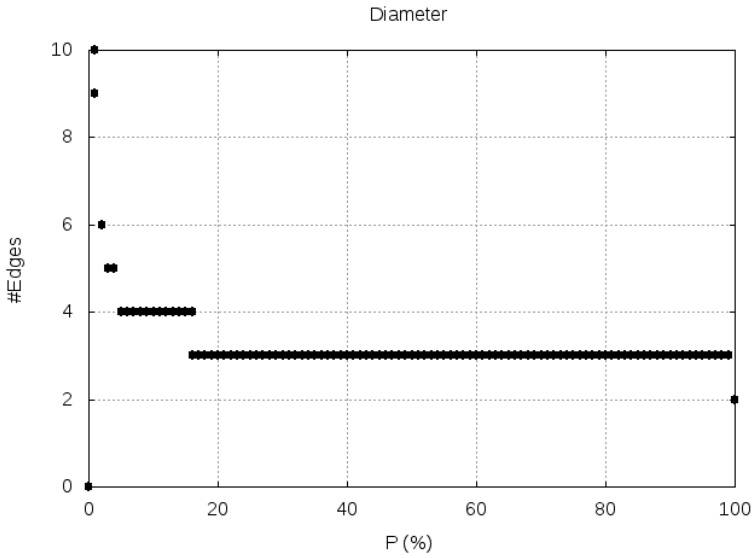

Figure 5.5: Diameter

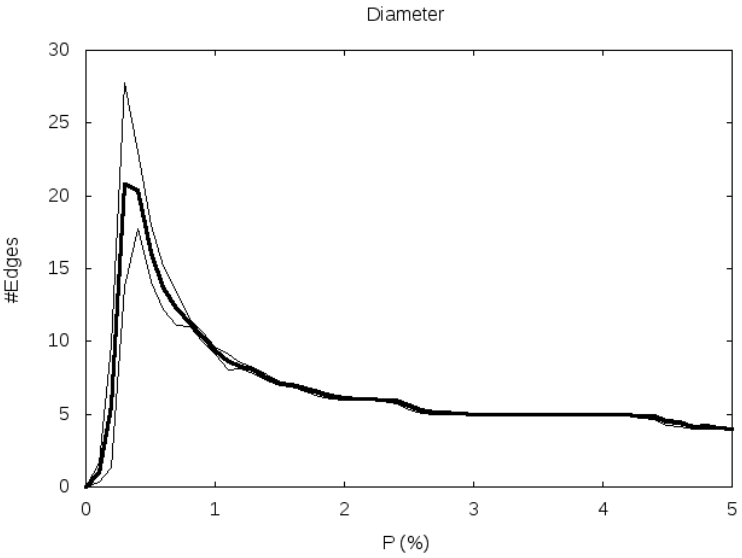

Figure 5.6: Diameter

5.4 Augmenting Paths per Iteration

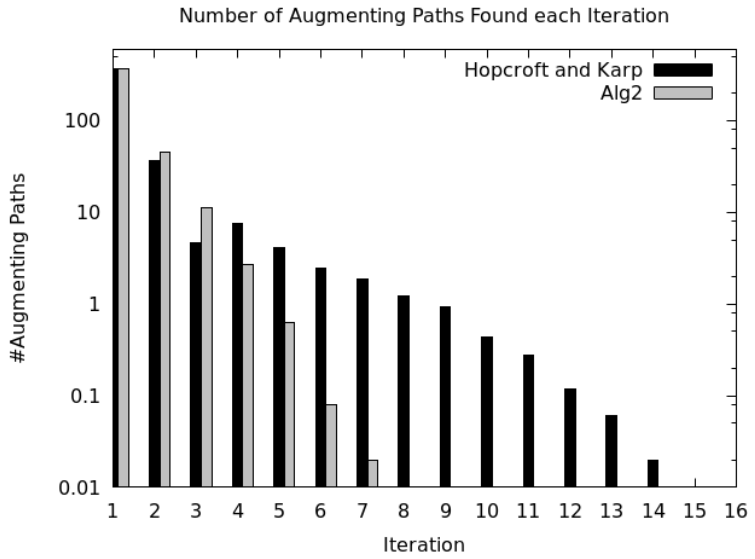

Figure 5.7: Number of augmenting paths found during each iteration with $p = 0.5$

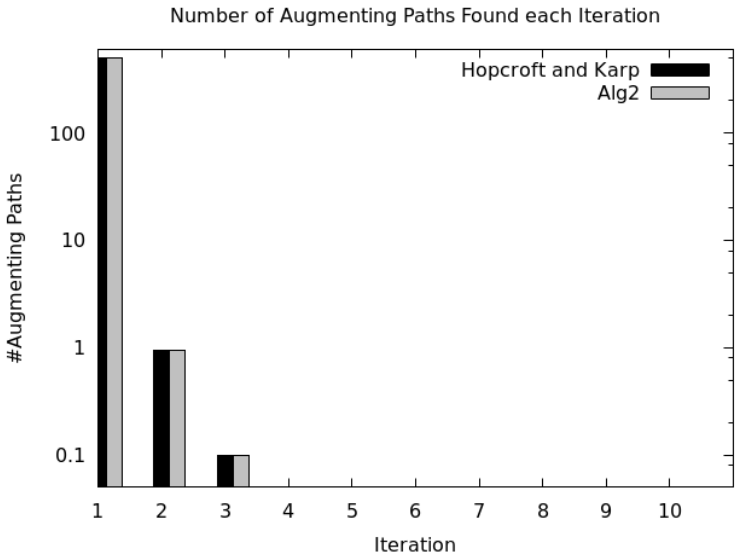

Figure 5.8: Number of augmenting paths found during each iteration with $p = 50$

In this section results of a small experiment are described to see how the number of augmenting paths found is distributed over the iterations done. This will explain some of the behaviour of the DFS.

In both Figures 5.7 and 5.8 can be seen that the most augmenting paths are found during the first iteration; during this iteration the biggest part of the matching is found. During the iterations thereafter, relatively very few paths are found: in the case of $p = 50$ in the second iteration on average 0.93 paths are found and in the third iteration 0.1 paths. For $p = 0.5$ in the second iteration there is an average of about 35 paths found, which is relatively still a very small amount compared to the 470 paths during the first iteration. It can also be seen that, even if the number of paths found is very small, H&K may require a significant larger amount of

iterations.

In Figure 5.7 it can also be seen that OH&K finds more augmenting paths in earlier iterations than H&K, while H&K might find augmenting paths in later iterations.

5.5 Comparing the Algorithms

5.5.1 p in the interval [0, 2%]

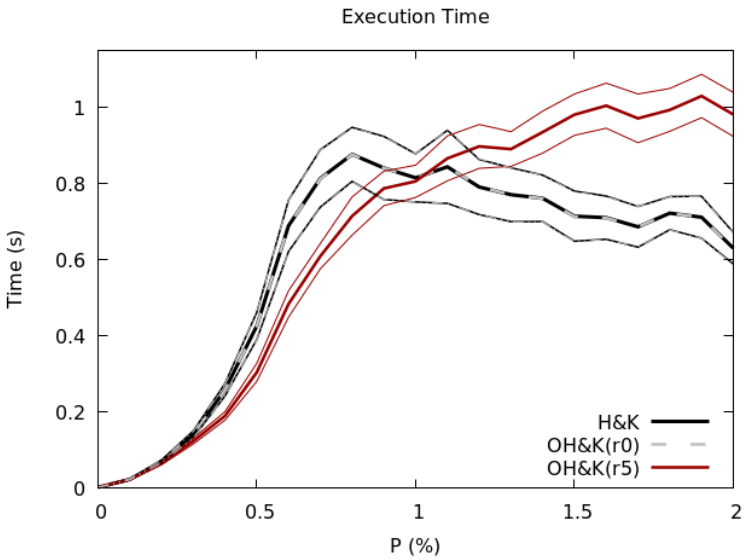

Figure 5.9: Execution time for random graphs with various values for p

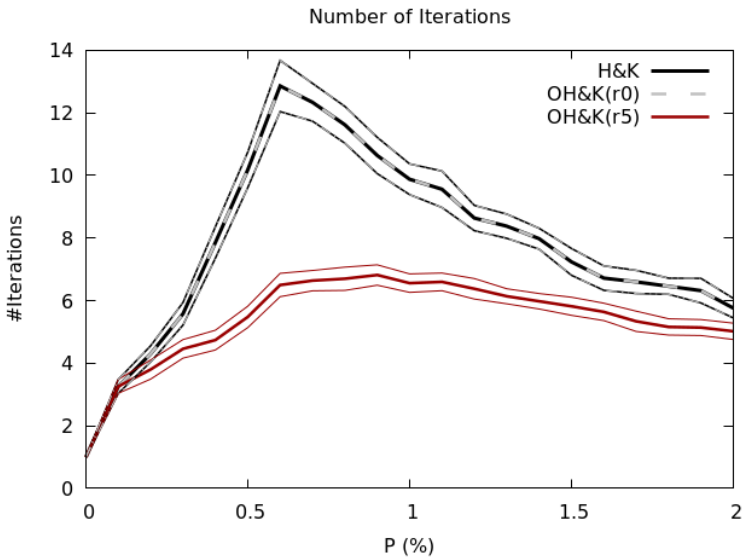

Figure 5.10: Number of iterations required for finding a maximal matching for random graphs with various values for p

To start, a look will be taken at all the parts of the algorithm and by using the previously found results, they will be explained.

In Figure 5.9 the execution time for the three algorithms can be seen with $\pm$ 3 times the standard deviation, which is shown in all figures in this section. The first thing to notice is that the results for OH&K with $r = 0$ and Hopcoft and Karp (H&K) overlap precisely. This can be seen in all the results in this and the next subsection. This means that by setting r to 0, OH&K will perform exactly the same as H&K.

It can also be seen in Figure 5.9 that the execution time rises for both algorithms until $p = 0.8$: after that, H&K requires less execution time and then it stays around the same point. The execution time for OH&K(r5), on the other hand, keeps on increasing after this point.

Comparing these results to the results in Figure 5.3, random graphs with $p = 0.8$ are very likely to contain a largest component containing (almost) all nodes in the graph. After that point, execution times of H&K seem quite stable.

The results cross when $p = 1.0$: which is the point where random graphs are very likely to contain a large

component containing most nodes in the graph as can be seen in Figure 5.3. It is also where there is a chance that random graphs contain a perfect matching according to 5.1.

It seems that according to these results, H&K performs more steadily and more efficiently on graphs consisting of one large component containing most nodes of the graph and with a chance to have a perfect matching; whereas OH&K performs better than H&K on random graphs that can not contain a perfect matching and the graph does not contain a large component that contains almost all nodes in the graph. These are also the random graphs with a larger diameter.

In Figure 5.10 it can be seen that in allmost all cases OH&K(r5) needs less iterations. For $p \in [0, 0.1\%]$ the graph contains a small amount of edges and both algorithms require the same amount of iterations. After that, the number of iterations H&K requires keeps on rising until $p = 0.6$. Then the number of iterations required by both algorithms seems to converge.

The idea of OH&K is that it can look further thus may require less iterations. For these random graphs, OH&K indeeds requires less iterations in most cases. However, despite that, OH&K is not always better than H&K regarding the execution time as shown in Figure 5.9. Which means that if the algorithms requires less iterations, even if it is a significant amount, it does not necessarily imply that its performance is better. To explain this behaviour, a look will be taken at the other steps of the algorithm. The steps, as discussed before, are:

1. (For OH&K) Determine a value for ℓ.
2. Generate the directed graph (BFS).
3. Find a maximal set of disjoint augmenting paths (DFS).
4. Compute the symmetric difference.

In the results below it can be seen that OH&K with $r = 0$ and H&K perform the same, thus determining a value for ℓ does not have a lot of influence on the performance of the algorithm, so we will not look any further at this step. During the second step, generating the directed graph, OH&K may look further than H&K, in which case OH&K may have to do more work. Thus a look will be taken at the number of edges visited during the BFS step and the size of all directed graphs generated. For the third step, finding a maximal set of disjoint augmenting paths, OH&K may have a larger directed graph than H&K, in which case OH&K may have to do more work. Therefore, results will be shown for the number of edges visited in the DFS step and they will be compared to the directed graph size. The fourth step, computing the symmetric difference, is about the same for both algorithms; since both algorithms will find a maximal matching, the number of times the symmetric difference between a matching and an augmenting path has to be computed is the same for both algorithms. However, when the sizes of paths may differ, this step might affect the performance. For this reason, a look will be taken at the total length of all augmenting paths found.

In Figure 5.11 it can be seen that around $p = 0.85$ the lines cross and after that H&K visits less nodes during the BFS than OH&K(r5). It is close to the intersection of the results for the execution time in Figure 5.9, but it

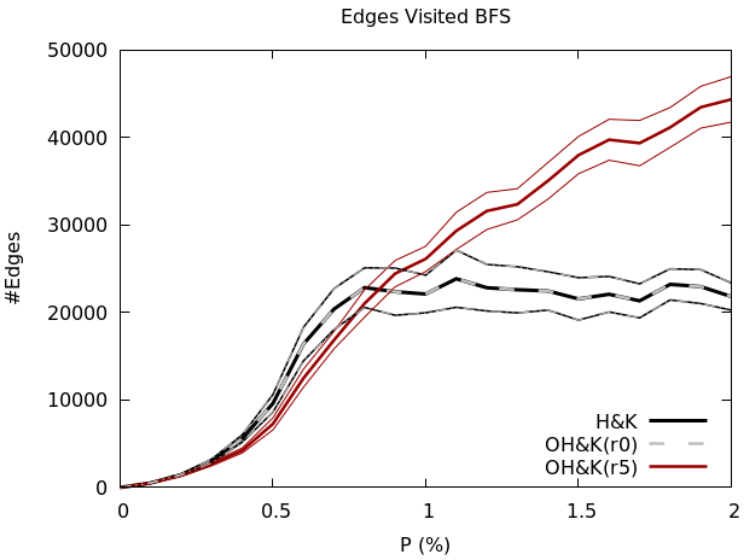

Figure 5.11: Number of edges visited BFS

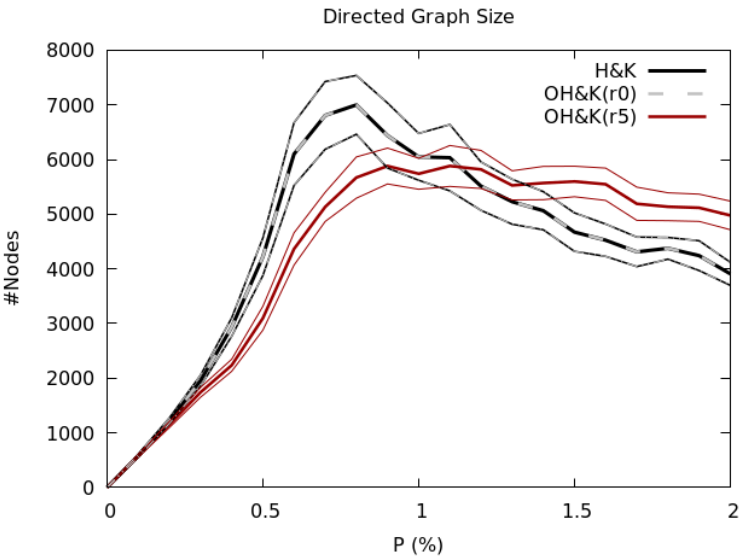

Figure 5.12: Size of directed graphs generated

is not exactly on the same value for p: in all cases where OH&K(r5) visits less nodes than H&K its execution time is smaller, but the converse is not always true.

After the intersection, the number of nodes visited by OH&K(r5) seems to grow almost linearly while the results for H&K seem to stay about the same. The results show a lot of similarity with the results for the execution time.

In Figure 5.12 the directed graph size is shown. The results intersect around $p = 1.1$, which is where the random graphs can have a perfect matching. Before the intersection OH&K(r5) needs less nodes for the graphs. When comparing this to the number of iterations, it can be concluded that OH&K(r5) does overall generate larger graphs since it requires less iterations and therefore generates less directed graphs. The number of nodes used for the directed graphs sligthly decreases after $p = 0.9$. For H&K it decreases faster and around $p = 0.9$. The difference in these results is of course because OH&K(r5) can look further for augmenting paths than H&K. Therefore it visits more edges for generating the directed graphs and the size of the generated directed graphs may be larger.

Since a different number of nodes is used for these directed graphs, the augmenting paths found by both algorithms may be different.

In principle, it is possible for the DFS algorithm to visit only those edges that are part of the augmenting path. This happens if all paths are found at once and no backtracking is required. In this case, also no paths are "cut off". In the initial graph there is always an augmenting path from one free node in B to another free node in A and during the DFS step only nodes are visited that are part of at least one augmenting path in the graph. However, since nodes and edges that are part of the maximal set of disjoint paths found so far are deleted from this directed graph, it is possible that this path no longer exists and no more augmenting path exists from that free node to the currently visited node to another free node: the augmenting path is said to be cut off and backtracking is required.

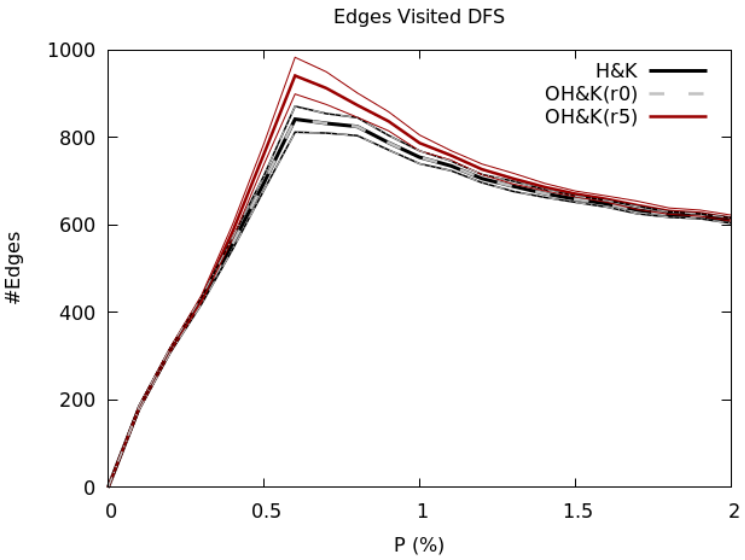

Figure 5.13: Number of edges visited DFS

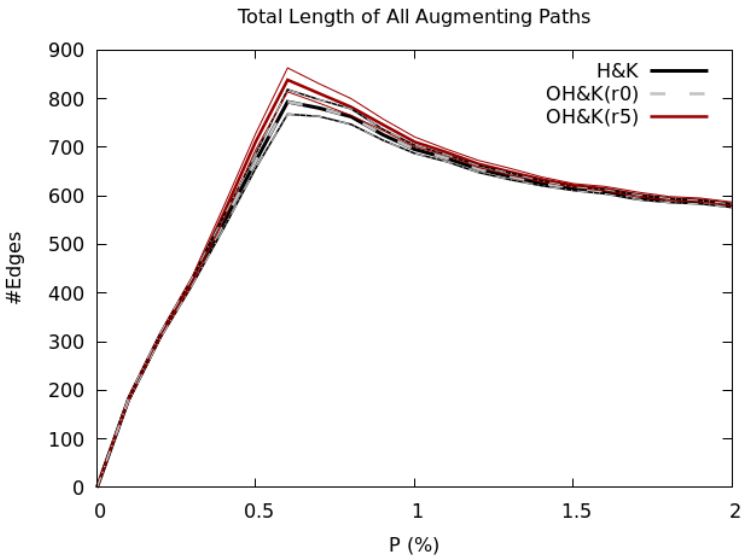

Figure 5.14: Total length of augmenting paths found

As seen in Figure 5.13 the number of edges visited for the DFS step decreases for both algorithms after $p = 0.6$. OH&K(r5) does visit less edges than H&K in general, but the difference in the results is not very large. It is also notable that even though p and thus the number of edges in the graph increases, the number of edges visited decreases, which may imply that the more edges in the graph, the less backtracking is done in this step; this is probably because if there are more edges, there are more possible augmenting paths so the chance that there actually is no augmenting path, containing both the free node from which the augmenting path starts and the current node visited, is smaller and less backtracking is required in general.

When comparing the results in Figure 5.13 to the results in Figure 5.14, it can be seen how much the number of edges visited in the DFS differs from the total length of all augmenting paths found. For H&K approximately at most 100 more edges are visited in the DFS step than the total length of the augmenting path, which means at least 100 possible augmenting paths are cut off and 100 times backtracking is required during this step. For OH&K(r5) this difference, which is approximately at most 50 smaller, less backtracking is required.

The difference between the total length of all augmenting paths found, like the number of edges visited during the DFS step, for both algorithms does not differ much.

To explain the differences in the performance of the algorithms it is thus important to look at the number of edges visited during the BFS step and the size of the directed graphs generated.

5.5.2 p in the interval [0, 100%]

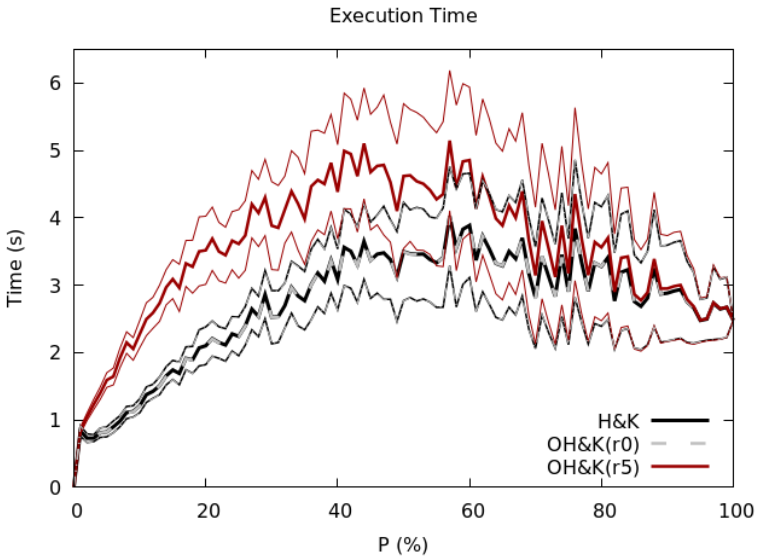

Figure 5.15: Execution time for random graphs with various values for p

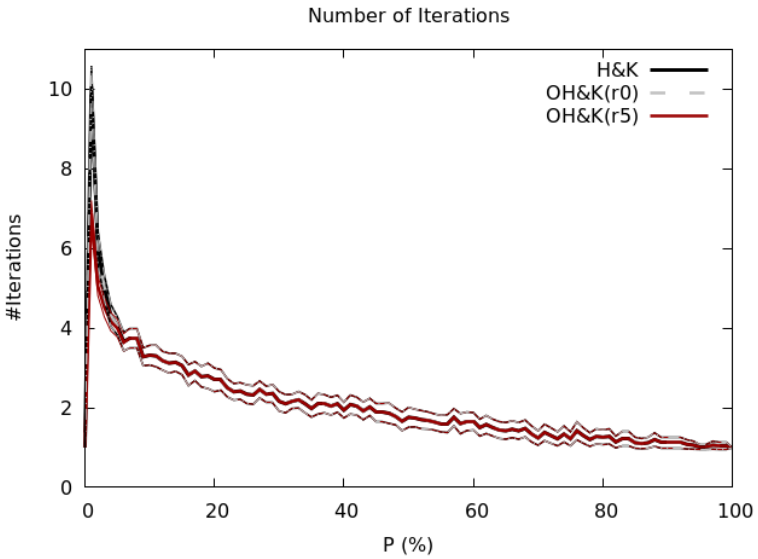

Figure 5.16: Number of iterations required for finding a maximal matching for random graphs with various values for p

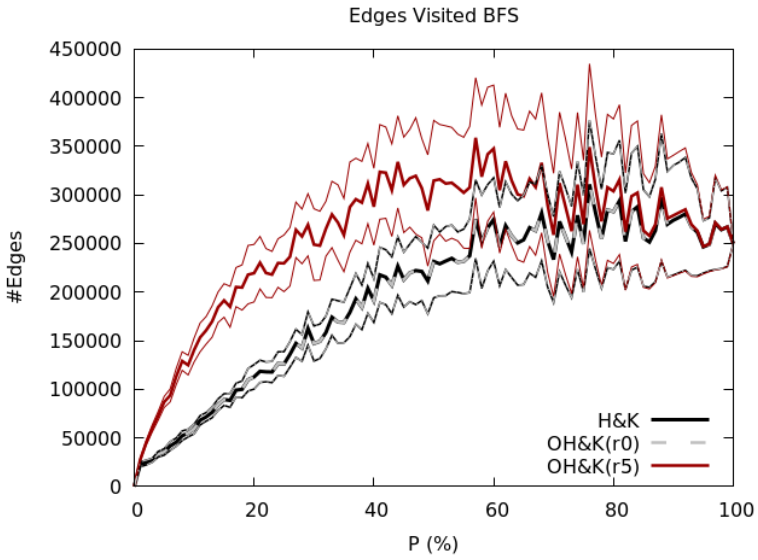

Figure 5.17: Number of edges visited BFS

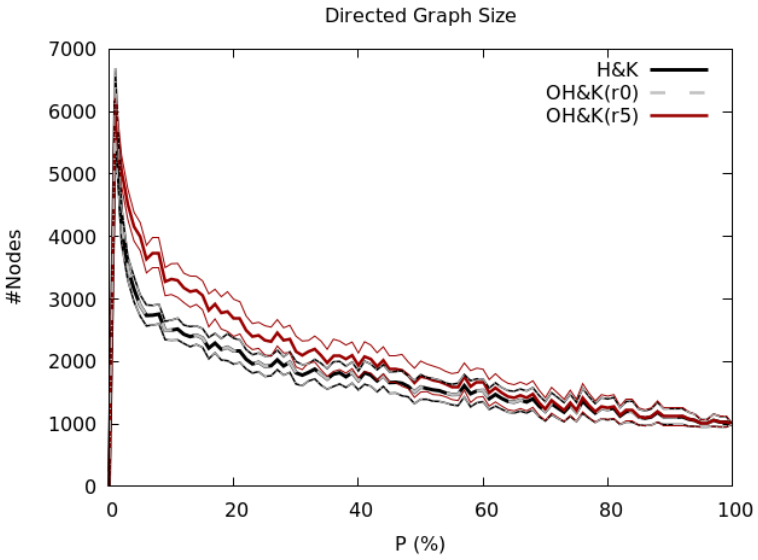

Figure 5.18: Directed graph size

In Figure 5.15 it can be seen that for p 0 to 100 with stepsize 1 OH&K(r5), is always less efficient than H&K. The reason why OH&K(r5) was more efficient in some cases for small values of p was that it required less iterations and therefore had to visit less edges in the BFS step. In Figure 5.16 it can be seen that both algorithms require the same number of iterations to find a maximal matching. Since OH&K is able to look further for augmenting paths than H&K, it always visits at least as many edges during the BFS step. In Figure 5.17 this difference is quite large, with as result that the performance of OH&K(r5) is significantly worse than H&K for larger values of p.

In Figure 5.18 it can be seen that OH&K(r5) also generates slightly larger directed graphs. Since the difference

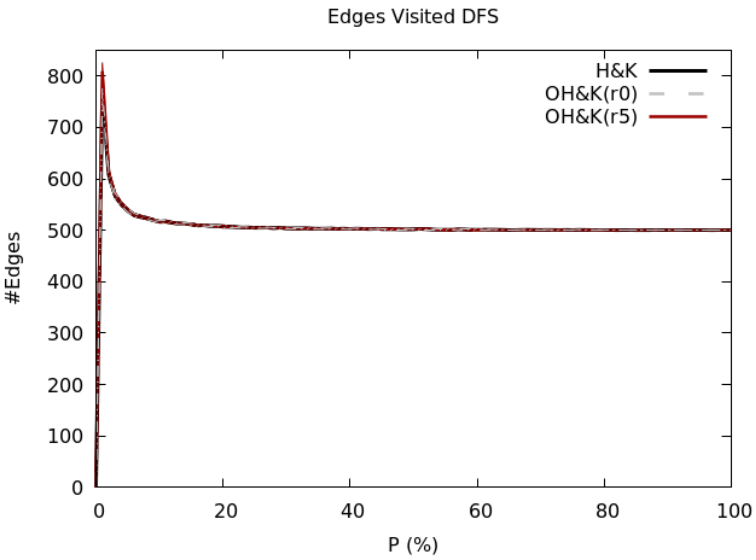

Figure 5.19: Number of edges visited DFS

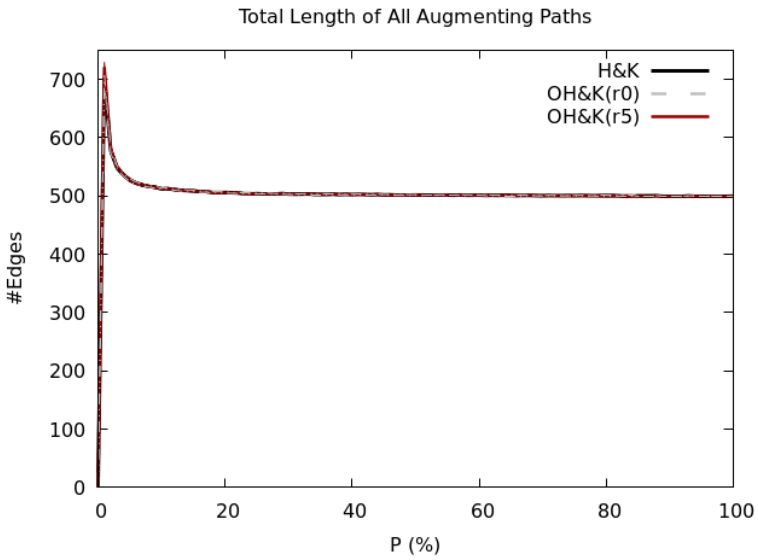

Figure 5.20: Total length of all augmenting paths found

between the nodes required for the directed graph and the number of visited edges is quite large, it may imply that OH&K(r5) visits a lot more edges that are not contained in the graph.

In Figure 5.19 it can be seen that the number of edges visited during the DFS step is approximately the same for both of the algorithms and according to Figure 5.20 the total length of augmenting paths found is approximately the same as well. Those results may imply that in most cases OH&K(r5) does not find paths that are longer. If it looks further than H&K, the generated directed graph is either very unlikely to contain more augmenting paths, or these paths, that are longer than the minimal length of the augmenting paths, are very rarely found.

5.6 Value for ℓ

The aim of this section is to compare various values of ℓ by setting r to different values. Since the previous section discussed which results explain the performance of the algorithms, only those will be shown in the current section. Since the results overlap quite a lot, plotting the graphs with the standard deviation would not clearly show the results. Therefore, the standard deviation is not plotted in the graphs in this section.

5.6.1 r in the interval [0, 5]

In Figure 5.21 it can be seen that until $p = 1$ OH&K($r > 0$) performs better than OH&K(r0). In that range OH&K(r5) and OH&K(r4) seem to perform the best. After that, OH&K(r0) performs better and its execution time decreases slightly.

The results of OH&K($r > 0$) converge such that only one line can be distinguished in the results, so their performance is approximately the same after $p = 1$ and their execution time keeps increasing. The results after

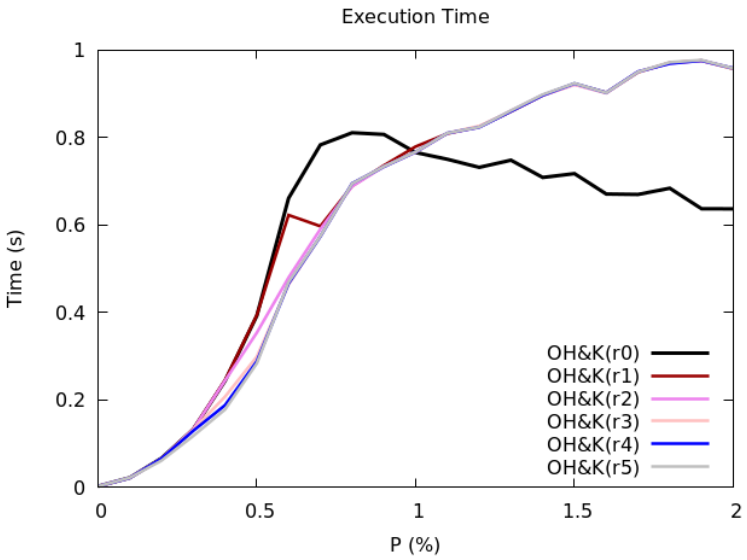

Figure 5.21: Execution time for random graphs with various values for p

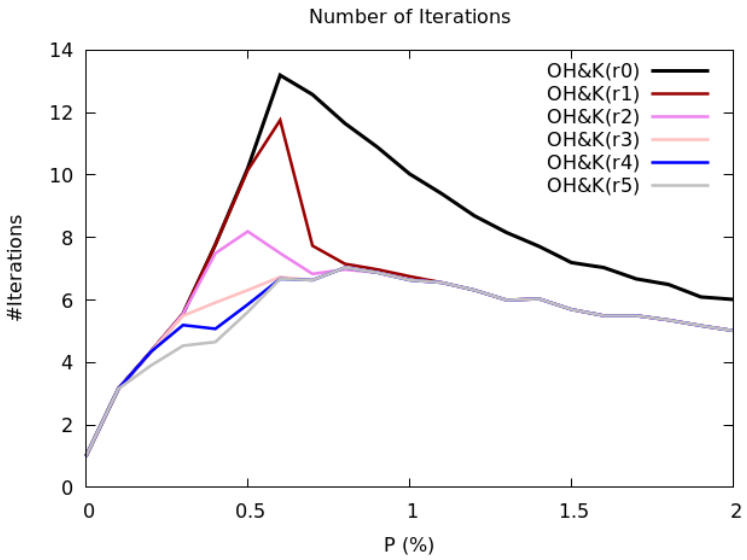

Figure 5.22: Number of iterations required for finding a maximal matching for random graphs with various values for p

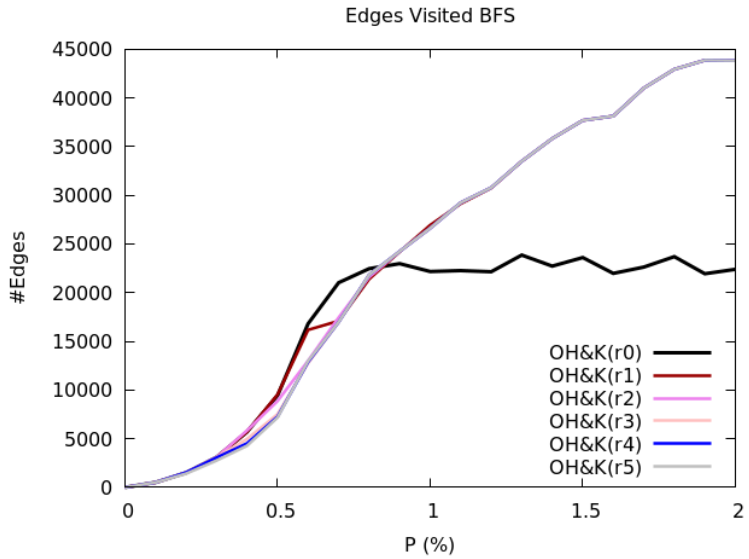

Figure 5.23: Number of edges visited during the BFS step

$p = 2$ are comparable to the results found in subsection 5.5.2 with OH&K(r5) representing OH&K($r > 0$).

Thus for values of p between 0 and 1, it is better to use a value for $r > 0$, while after $p = 1$ setting r to 0 ensures a better performance in those cases.

In Figure 5.22 the number of iterations required for each algorithm can be seen. As OH&K(r5) can look the furthest of all algorithms, that algorithm requires the smallest number of iterations overall. However, despite the large difference in the number of iterations required, the difference in execution time, as seen in 5.21 is not that significant. This is because, as shown in Figure 5.23 the difference in the number of edges visited in the BFS step is not as large, despite the difference in the number of iterations required.

5.6.2 r in the interval [0, 1]

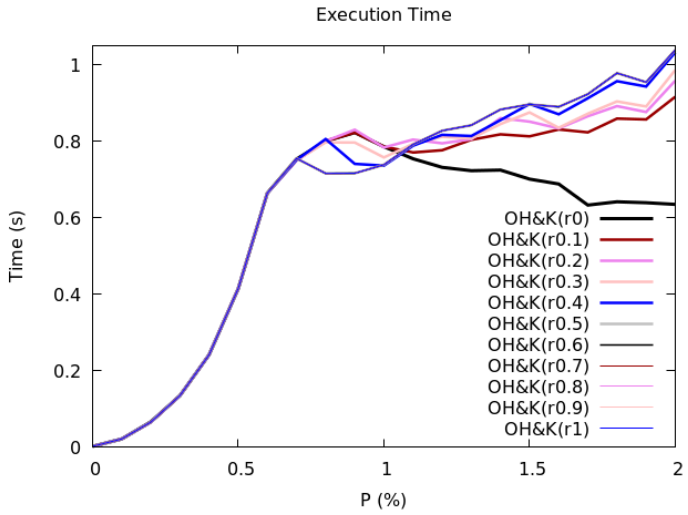

Figure 5.24: Execution time for random graphs with various values for p

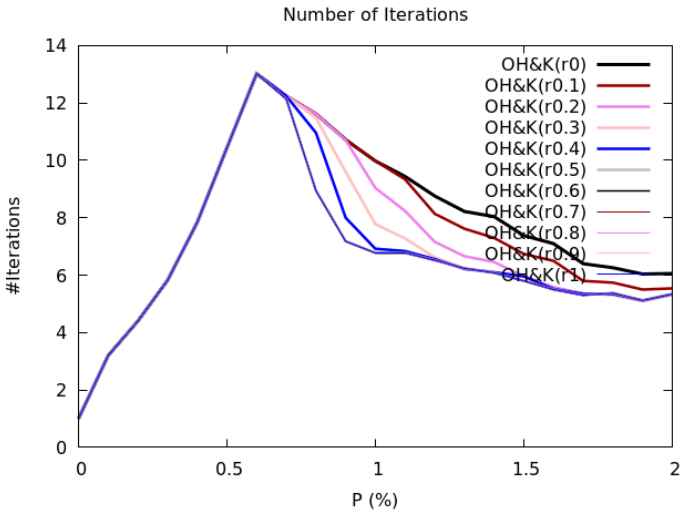

Figure 5.25: Number of iterations required for finding a maximal matching for random graphs with various values for p

In the results in Figure 5.24 can be seen that using values for r that are smaller than 1, the results do not differ a lot in the range of $p = 0$ to $p = 1$. After $p = 1$ the results of OH&K($r > 0$) do differ more with the algorithms with a smaller value for r having a better performance.

According to the results in Figure 5.25 the number of iterations required by the algorithms is the same in the range for $p = 0$ to $p = 0.6$. After that, a difference occurs: the larger the value for r, the fewer iterations are required. The results also seem to converge after $p = 1.5$.

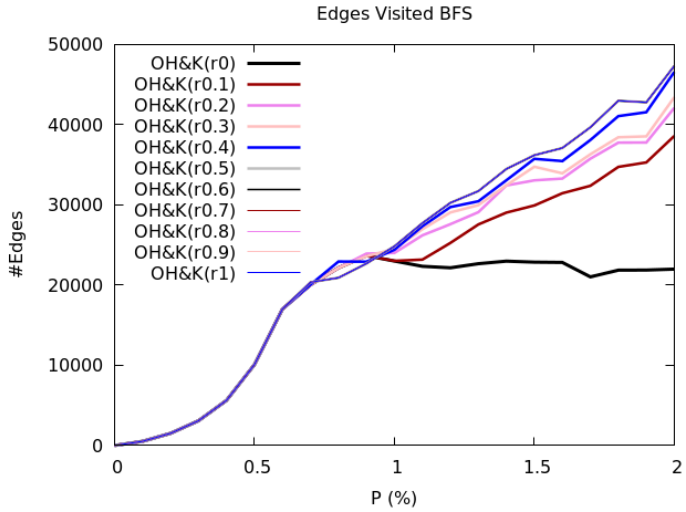

Figure 5.26: Number of edges visited during the BFS step

In Figure 5.26 it can be seen that until $p = 1$ the number of edges visited during the BFS step are approximately the same. However, after $p = 1$ the results clearly diverge with OH&K($r0$) visiting less edges than OH&K($r > 0$) and the lower the value of r, the smaller the number of edges visited.

Chapter 6

Conclusions

The algorithms have been compared and discussed and various different formulas for ℓ have been tested. The most important found results are the following:

According to the results found when looking at all possible values p ($p = 1$ to $p = 100$) for random graphs, the algorithm of Hopcroft and Karp performs better. However, when looking at a lower value of p ($p = 0$ to $p = 1$) Optimistic Hopcroft and Karp performs better for all values of ℓ tested where $r > 0$.

In subsection 5.5.1 has been shown that when setting the value of r to 0, Alg2 behaves the same as the algorithm of Hopcroft and Karp.

In chapter 5.6 can be seen that when using a high value for r, and thus for ℓ, the algorithm performs better for random graphs with $p < 1$. When the value of p is larger, the algorithm would perform better if r, and thus ℓ, is set to a lower value.

Therefore it could be said that, if the value of ℓ can be chosen optimally, Optimistic Hopcroft and Karp performs at least as well as the algorithm of Hopcroft and Karp. However, it is yet unknown how to choose this value optimally.

To answer the research question: ”Which algorithm performs better?” The answer would be that it depends on the graph and how ℓ is chosen: Alg2 could perform significantly worse than the algorithm of Hopcroft and Karp, and in some cases it could perform significantly better.

Bibliography

[Ber57] Claude Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences of the United States of America*, 43(9):842, 1957.

[Blu90] Norbert Blum. A new approach to maximum matching in general graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 586–597. Springer, 1990.

[CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

[ER60] Paul Erdös and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[ER66] P Erdős and Alfréd Rényi. On the existence of a factor of degree one of a connected random graph. *Acta Mathematica Hungarica*, 17(3-4):359–368, 1966.

[FK15] Alan Frieze and Michał Karoński. *Introduction to random graphs*. Cambridge University Press, 2015.

[Gre06] Brian Green. Edmondss non-bipartite matching algorithm (lecture notes). *U.C. Berkeley*, February 2006.

[HK73] John E Hopcroft and Richard M Karp. An n^5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.

[KUW86] Richard M Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random nc. *Combinatorica*, 6(1):35–48, 1986.

[Les99] Jure Leskovec. Snap for c++: Stanford network analysis platform, 1999.

[MV80] Silvio Micali and Vijay V Vazirani. An o (v— v— c— e—) algoithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*, pages 17–27. IEEE, 1980.

[TK11] Frank W Takes and Walter A Kosters. Determining the diameter of small world networks. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1191–1196. ACM, 2011.