INRIA: "The French Institute for Research in Computer Science and Automation"
Team COATI: "Combinatorics, Optimization, and Algorithms for Telecommunications"

Research Report

k-Maximum Matching and Applications

Theodoros Alexandros KARAGKIOULES

Supervisor:
Nicolas NISSE
Research Officer
Inria

Contents

Introduction

This work is the result of a 6 months internship, under the supervision of research Officer Nicolas NISSE.

The original scope of the internship was the introduction to the k-maximum matching problem along with the application of a special case of the problem on a industrial-specific variant, and the theoretical exploration of the k-maximum matching problem on specific graph families.

The Internship was divided into two independent stages that ran simultaneously.

On the first stage of the internship, inspiration was drawn from an application of the k- maximum matching on airport slot assignment. As it's described in the following sections the k-maximum matching problem can be used for the reassignment of flights to time slots for landing. In collaboration with Amadeus, the k-maximum matching algorithm was applied on actual data to test its efficiency.

During the second phase of the internship, all the combinatorial optimization aspects of the k-maximum problem (complexity etc) were introduced. The case where $k = 3$ was studied and the proposed algorithm [7] for that case was implemented and applied on random graphs and on bipartite graphs that simulated the Amadeus data.

Additionally, based on a proposed algorithm for Paths [7] a new algorithm was proposed that computes the k-maximum matching on a specific type of tree, the Spider.

CHAPTER 1

Algorithms

1.1 Maximum matching

In this section the Classical Maximum matching is presented.

Let $G = (V, E)$ be a graph. A *matching* $M \subseteq E$ of G is a set of edges pairwise disjoint. A vertex $v \in V$ is defined as *covered* by M if there is $e \in M$ such that $v \in e$. Otherwise, v is said *exposed*. The size of a maximum matching in G is denoted by $\mu(G)$.

The problem of computing a maximum matching has been widely studied and it is well known that it can be solved in polynomial-time. The most popular algorithm for computing the classical maximum matching is by Edmonds[4]. This algorithm has been used broadly in the calculations on random graphs that follow in the next chapter.

Augmenting Paths A path $P = (v_0, \cdots, v_k)$ of G is a sequence of pairwise distinct vertices such that $e_i = \{v_i, v_{i+1}\} \in E$ for each $0 \leq i < k$. The path P is said M-augmenting if, v_0 and v_k are exposed and, for any $0 \leq i < k$, $e_i = \{v_i, v_{i+1}\} \in M$ if and only if i is odd. In particular, an augmenting path has always an odd length. A well known theorem of Claude Berge states that a matching M is maximum if and only if there are no M-augmenting paths [1].

Related work The first algorithm for solving the maximum matching problem in polynomial-time in general graphs is due to Edmonds [4]. Then, many work has been dedicated to design more efficient algorithms . In particular, the algorithms in [5, 6] are based on augmenting paths in the non-decreasing order of their lengths.

1.2 k-maximum matching

Let $\mu_k(G, M)$ denote the size of a maximum matching that can be obtained from $M \subseteq E$ in G by augmenting paths of length at most $kk \geq 1$, where k is an odd integer. To define the k-*Matching Problem* one must compute a sequence of augmenting-paths of length $\leq k$ that allow to obtain, from M, a matching of size $\mu_k(G, M)$.

In the cases $k \in \{1, 3\}$, it has been proven that the computational complexity of the k-Matching Problem is equivalent to the one of the classical maximum matching problem (without any constraint on the length of paths). Hence, for $k \in \{1, 3\}$, it can be solved in polynomial time.

1.2.1 k-matching for Airport Slot assignment

The k-maximum matching problem has been found to have an application in the aviation industry. When an aircraft is approaching an airport, it is assigned by Air Traffic Control a landing time (called *slot*). If there is a delay for any reason (weather conditions, late arrival, priority), it looses its slot and a new slot needs to by reassigned to it. However, slots for landing are a scarce resource of the airports and, to avoid conflicts and for safety reasons, Airline Operation Controllers have to regularly modify the assignment of the slots of the aircraft.

This modification of the slot-assignment uses only two kinds of operations: either assign aircraft A to a free slot, or give A the slot of B and assign B to a free slot. The problem is then the following:

Let $k \geq 1$ be an odd integer and let G be a graph and M be a matching of G. What is the maximum size of a matching that can be obtained from M by using only augmenting paths of length at most k?

Let $G = (X \cup Y, E)$ be a bipartite graph. X represents the set of aircraft and Y represents the set of available slots. There is an edge between $a \in X$ and $s \in Y$ if the slot s is compatible with the schedules of aircraft A. Let M be a matching of G that corresponds to a pre-established valid assignment of some slots to some aircraft . The problem of reassignment of slots is equivalent to computing a maximum matching that can be obtained from M by augmenting only paths of length at most 3.

1.2.2 The case where k=3

Let G be a graph and $M \subseteq E(G)$ be a matching of G. An edge $e \in E(G) \setminus M$ is said *useless* if e is incident to two edges of M, i.e., if each endpoint of e is incident to one edge of M. An edge $e \in M$ is said *forced* if at least one of its endpoints has degree one in G.

Finally, let $\hat{G}$ be the graph obtained from G and the matching M by first removing all useless edges (but keeping their endpoints), and then by removing the forced edges and their endpoints. Let $\hat{M}$ be the matching resulting from M in $\hat{G}$, i.e., $\hat{M}$ equals M minus the forced edges. Note that $\hat{M}$ has no useless edges nor forced edges. Abusing the notations, we will identify the paths in $\hat{G}$ with some paths in G.

This algorithm has been used to implement the k-maximum matching programme that computes the data provided by Amadeus and also the randomly generated graphs of the following chapter.

1.2.3 The case of Paths

Let $Q = (v_1, \cdots, v_i)$ be the maximum alternating path containing v_1. If Q is not augmenting or if Q has length more than k (i.e., $i > k$), the algorithm is applied recursively on $P \setminus (v_1, \cdots, v_{i-1})$. Otherwise, Q is augmented (it is added to the desired sequence of paths) and the algorithm is applied recursively on $P \setminus Q$.

Algorithm 1 k-matching(P_n, M)

```
Require: A path $P_n = (v_1, \cdots, v_n)$, a matching $M \subseteq E(P)$ and an odd integer $k \geq 1$.
Ensure: A sequence $(Q_1, \cdots, Q_r)$ of augmenting paths with $r = \mu_k(P, M) - |M|$.
1: if $n = 1$ then return $\emptyset$
2: else
3:     Let $1 < i \leq n$ be the smallest integer (at least 2) such that $v_i$ is exposed.
4:     if $v_1$ is not exposed OR $i > k$ then
5:         Let $P' = (v_i, \cdots, v_n)$ and $M' = M \cap E(P')$. return k-matching($P', M'$)
6:     else
7:         Let $P' = (v_{i+1}, \cdots, v_n)$ and $M' = M \cap E(P')$. return $\{(v_1, \cdots, v_i)\} \cup$ k-matching($P', M'$)
8:     end if
9: end if
```

1.2.4 The case of the Spider

Theorem 1.1. *Let S be a spider and M its initial matching, and $(P_1, \ldots, P_r)$ a sequence of augmenting paths of length $\leq k$, such that $\mu_k(S, M) = |M| + r$ then $\exists$ $(Q_1, \ldots, Q_r)$ a sequence of augmenting paths of length $\leq k$ such that at most one such path contains the center C.*

Proof. Let S be a spider, and lets assume that the optimal sequence of augmenting paths SP is known along with the set of branches Br (those paths of S from a leaf to the nn of C.
Assume for contradiction that $\exists \geq 2$ augmenting paths that contain the center C and let P_i and P_j be the first such paths. Let $P_i = (u_{m_i}, \ldots, C, \ldots, u_{n_i})$ and $P_j = (u_{m_j}, \ldots, C, \ldots, u_{n_j})$ and let $Q = (u_{m_i}, \ldots, C, \ldots, u_{n_j})$ be the **(augmenting?)** path containing partially P_i, the center C and partially P_j. Since $[C, u_{n_i}] \subset [C, u_{n_j}]$ in order for P_j to be an augmenting path, then $\exists$ $H = (P_j \setminus P_i) \setminus Q$. Let's also assume that there is a set of paths $A = \{A_1, \ldots, A_t\} \in SP$ between P_i and P_j and $A \subseteq Br$, such that $\forall k \in \{1, 2, \ldots, t\}$ $A_k \setminus P_i = \emptyset$ or $A_k \setminus P_j = \emptyset$ .Also there is a set of paths $B = \{B_1, \ldots, B_f\} \in SP$ between P_i and P_j and $B \subseteq Br$, such that $\forall n \in \{1, 2, \ldots, f\}$ B_n is non intersecting with any paths in A,Q, P_i, P_j and H. Then the optimal sequence of augmenting paths can be the following:

$P_1, P_2, P_3, \ldots, A_1, \ldots A_t, Q, B_1, \ldots, B_f, H, \ldots, P_r$, where augmenting only path Q once can lead to the equivalently optimal solution. □

Algorithm 2 Spider algorithm(S,M)

```
 1: $M' = M$
 2: Let there be a set of leaves $\{\lambda_1, \ldots, \lambda_r\}$
 3: $x_k$ is the nn of $C$ in the path $C \rightarrow \lambda_k$
 4: for $i \in \{1, \ldots, r\}$ do
 5:     for $j \neq i$ do $M''$=Path_Algorithm($\lambda_i \rightarrow \lambda_j, M$)
 6:         for $k \notin \{i, j\}$ do $M''$=$M''$∪Path_ Algorithm($\lambda_k \rightarrow x_k, M$)
 7:         end for
 8:     end for
 9:     if $M'' > M'$ then
10:         $M' = M''$
11:     end if
12: end for
13: return $M'$
```

Proof. Let there be an optimal solution that is obtained by the paths in the sequence $Q' = (P'_1, \ldots, P'_r)$ for the spider.

Let there, also, be a set of paths from leaf to leaf $PLL = \{Pll_1, \ldots, Pll_n\}$, a set of branches $BR = \{Br_1, \ldots, Br_m\}$ and a sequence of augmenting paths $Q = (P_1, \ldots, P_r)$, such that for $j \in \{1, \ldots, r\}$ $P_j \in PLL$ and $\forall\ i \neq j \in \{1, \ldots, r\}$ $P_i \in BR$.

Applying the path algorithm for P_j and then for the $r - 1$ paths of Q and due to **Theorem 2.1** it is ensured that the matching produced will lead to a feasible solution, since C is never exposed.

Due to the iterative nature of the algorithm it can be checked that all possible combinations of P_j and P_i that construct Q, are considered and that one of the solutions computed as above will be the initial optimal solution Q'.

Thus by induction Spider Algorithm will always produce an optimal matching on a Spider. □

Chapter 2

Application

2.1 Description

This chapter aims at providing the picture of analysis on the data provided by Amadeus the bipartite graphs and the random graphs generated.

The random graphs have been generated as purely random graphs based on the Reyni Erdős and the Barabási Albert with different values of connectivity probability or connectivity.

The bipartite graphs were generated on the density of the Amadeus graphs to simulate better the results. Additionally an extra set of test have been made on bipartite graphs that simulate the degree distribution of the Amadeus graphs. Certain classes of degree distribution were selected and based on those 480 bipartite graphs have been generated.

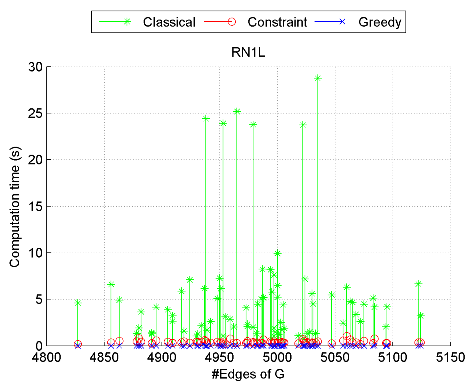
At every graph the classical matching is computed based on the Edmonds algorithm, then the k-matching algorithm is used as described in section 1.2.2. Lastly a greedy algorithm is implemented to be used for comparison.

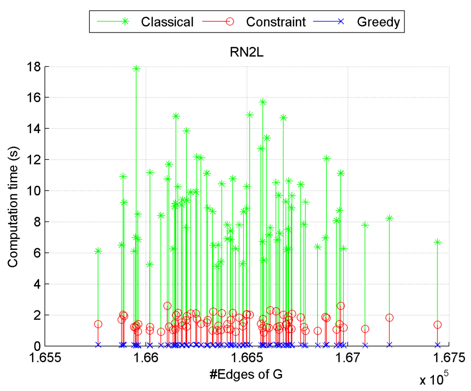
The ultimate goal is to explore the efficiency of the k-matching algorithm for the case of $k = 3$ and to compare it with the result of the Edmonds algorithm and of course the Greedy implementation.The results follow along with some comments.

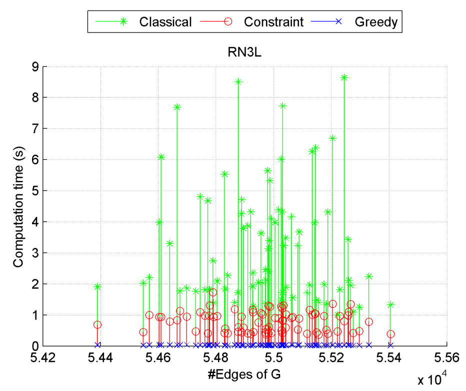
2.2 Reyni Erdős

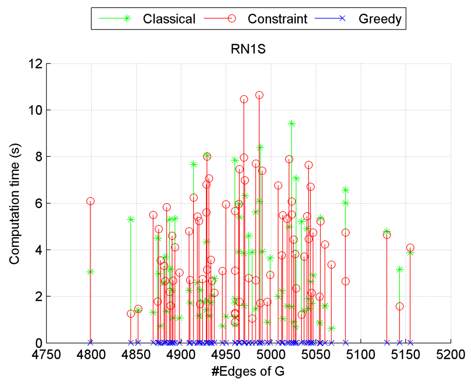
- RNxy are randomly generated graphs with a fixed number of 1000 nodes
- There are 6 categories based on the initial matching and probability p of connection
- $x = \{L, S\}$, L When the initial matching is large and S when it is small
- $y = \{1, 2, 3\}$ 1 when $p = \frac{\log n}{n}$, 2 when $p = \frac{1}{3}$ and 3, when $p = \frac{\log n}{n} + 0.1$

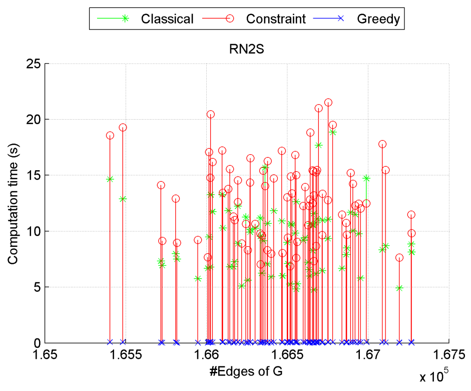
Computation time over $|E|$

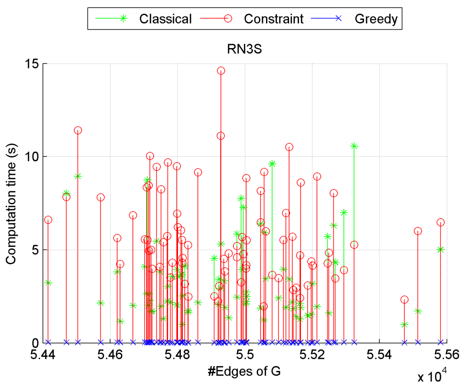










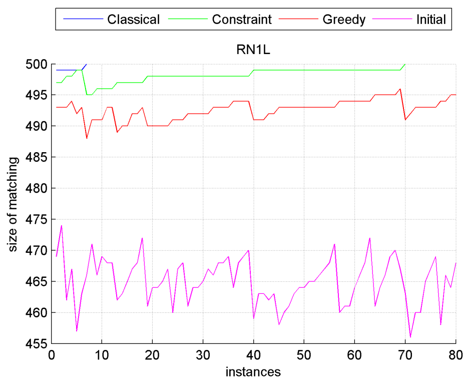


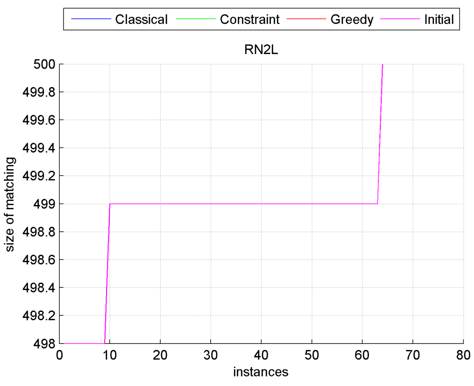
Computation time over $|E|$ - Comments

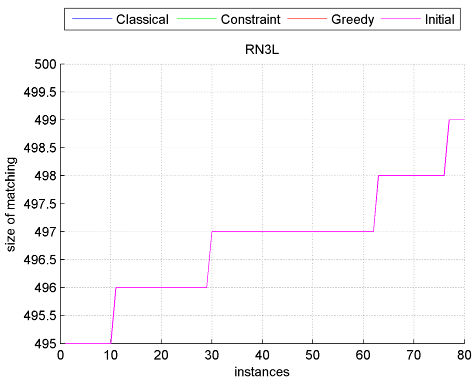
- This is a very straightforward set of figures

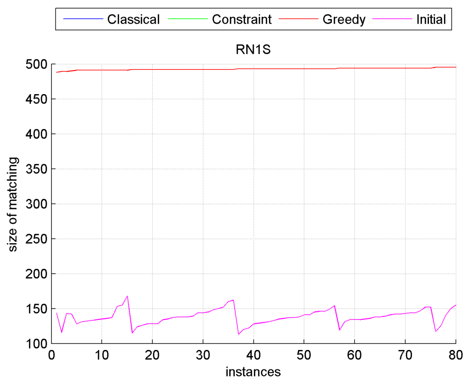
- When the initial matching is sufficiently large then the forced edges are taken out(a process which takes a fragment of the computation time) and then the classical matching algorithm (Edmonds) is applied on a smaller graph($\hat{G}$). This reduces significantly the computation time.
- When the initial matching is small then the computation time of the constraint algorithm might be larger than that of Edmonds. The reason is that the constraint algorithm need some time to compute the useless and forced edges and then Edmonds algorithm is applied on $\hat{G} \simeq G$. Thus the overall processing time is sometimes larger.

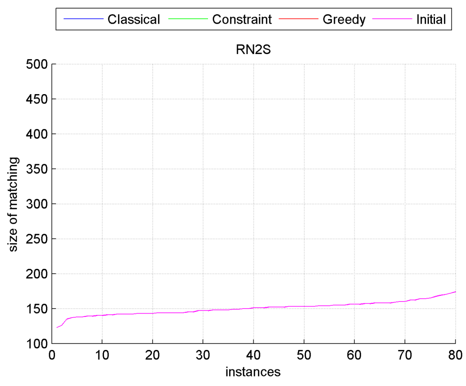
Sorted matching size

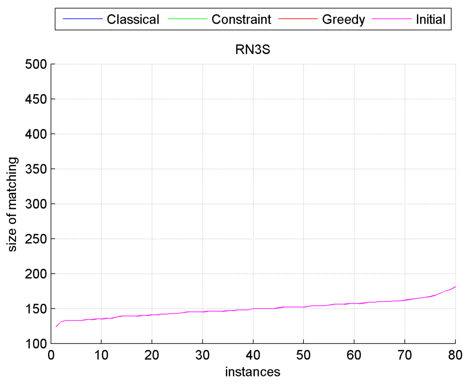












Sorted matching size - Comments

- This set of figures represents the ascending order of the Classical matching size along with its corresponding Constraint and Greedy equivalent.

- When the probability is sufficiently higher than the threshold $\frac{\log n}{n}$ then the three algorithms produce identical results of optimal matching.

- For lower probability there is a difference between the results of the algorithms

- the degree of that difference is depicted in the first set of figures presented.

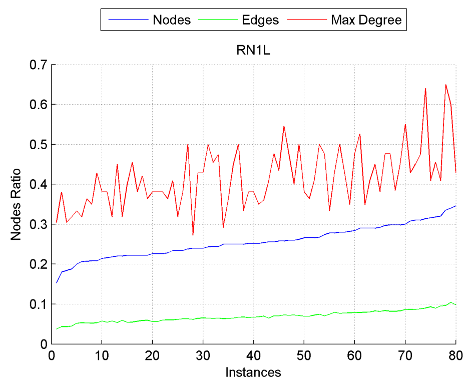
- It is interesting to observe that a change on the Constraint algorithm is associated with a even more significant change in the Greedy algorithm.

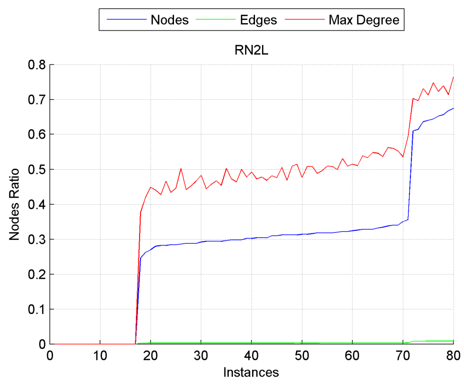
- There is a difference only when the probability is at the connectedness threshhold $\frac{\log n}{n}$

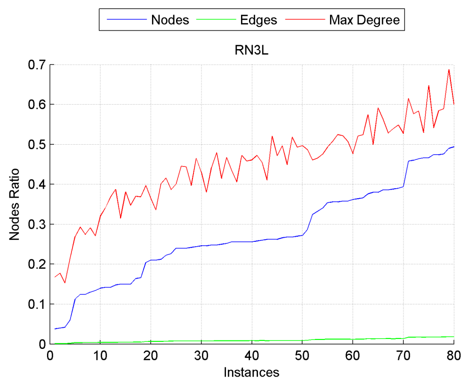
- The difference is very small compared to the size of the matching ($\simeq$ 500)

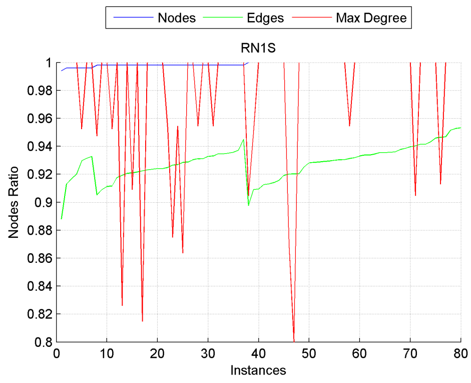
- The difference in matching size appears mostly at the Greedy implementation when compared to the constraint algorithm

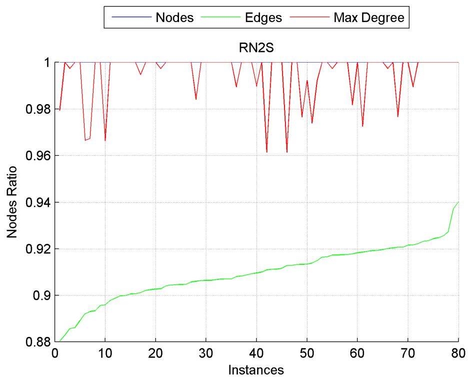
Sorted Ratios

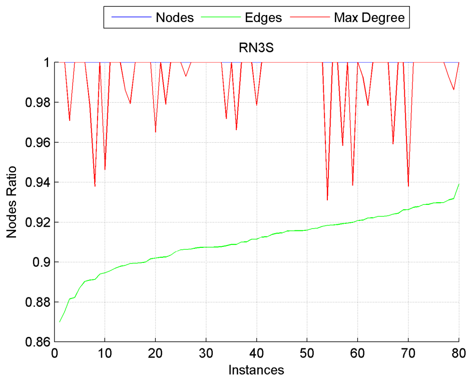








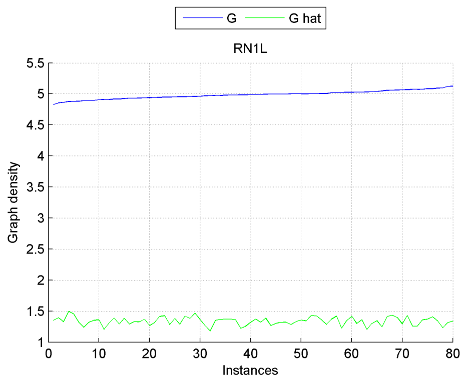


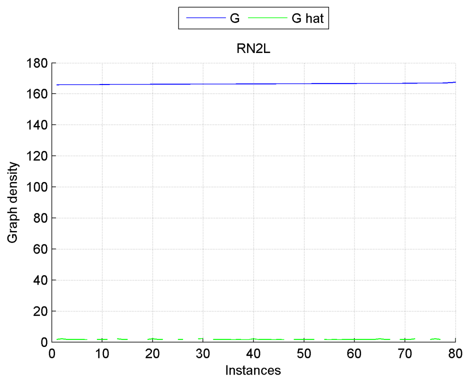


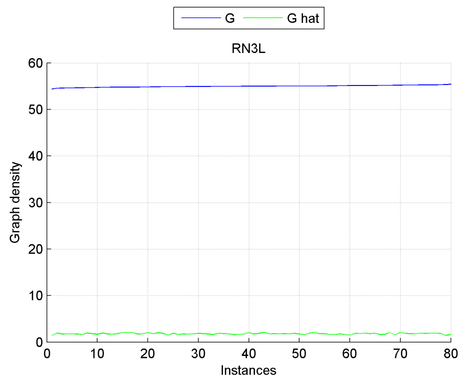
Sorted Ratios - Comments

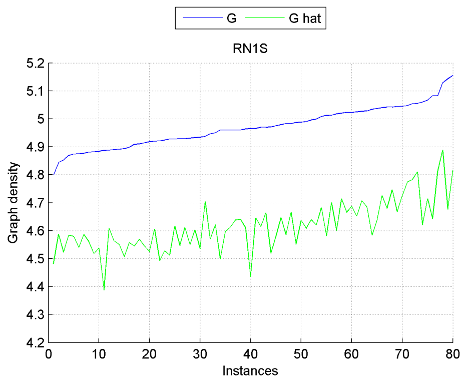
- This set of figures depicts the ratios: $\frac{\hat{V}}{V}$, $\frac{\hat{E}}{E}$, $\frac{\hat{\Delta}}{\Delta}$,in ascending order of $\frac{\hat{V}}{V}$
- It is evident that when the initial matching is large the amount of forced edges (and their endpoints) removed is high, thus the ratio of Nodes and edges is very low. For higher probabilities almost all edges are removed as useless.
- For a smaller initial matching almost no forced edges (and their endpoints) are removed which we conclude be the node ration being close to 1. There is some small amount of useless edges removed though since the ratio of edges is less but close to 1
- The ratio of Δ for large initial matching follows the trend of the node ratio, yet for smaller initial matching it deviates significantly.

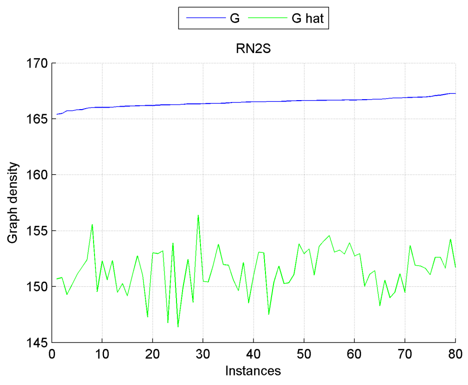
Sorted $\frac{|E|}{|V|}$

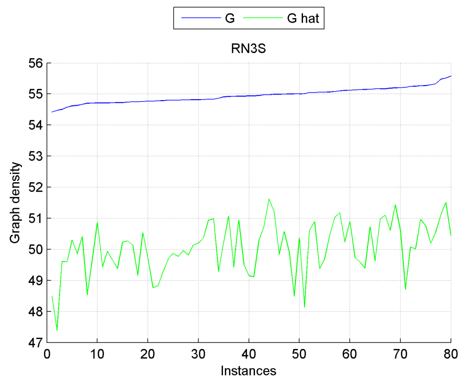










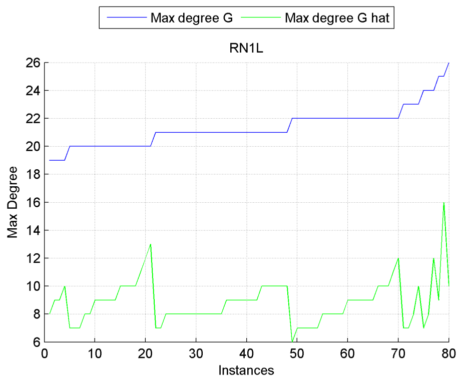


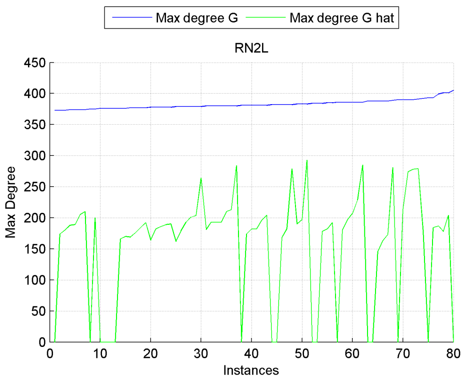
Sorted $\frac{|E|}{|V|}$ - Comments

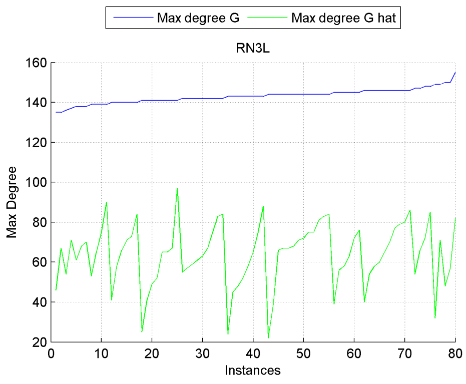
- Although as graph density the ratio $\frac{2|E|}{|V|(|V|-1)}$ is considered, in this set of figures we evaluate the correlation of the ratio $\frac{|E|}{|V|}$ of G and $\hat{G}$, since the density of $\hat{G}$ could acquire values higher than that of G
- The figures are calculated in ascending order for the ratio of G
- For the case of large initial matching the two ratios for G and $\hat{G}$ remain almost constant (to the extent of the probability of each graph), since removing forced edges(and their endpoints) alters the $|E|$ and $|V|$ proportionately.

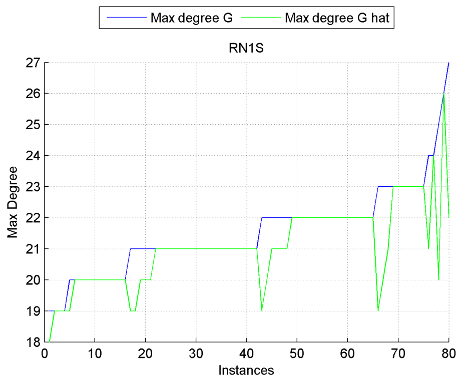
- For the case of small initial matching we observe some deviation in the values of the ration for $\hat{G}$, and that is explained with the removal of only useless edges which only alters the $|E|$ and keeps $|V|$ in the original values.

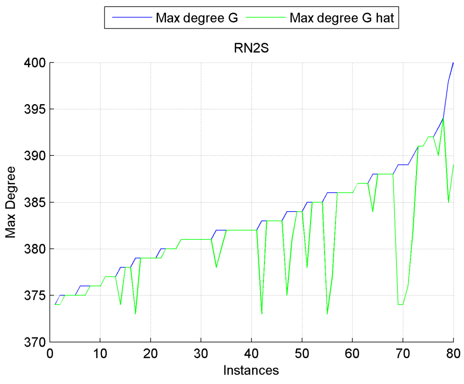
Sorted Δ

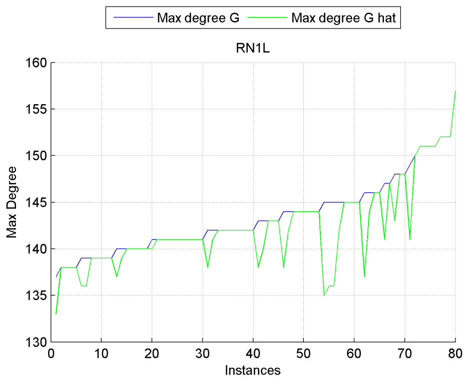












Sorted Δ-Comments

- In this set of figures the ascending order of the Δ of G is associated with the Δ of $\hat{G}$
- There is no significant jump of the Δ for each graph category which tells that that we are comparing a uniform distribution of random grapghs for each probability case

- For larger initial matching we observe the difference in the Δs of G and $\hat{G}$ given the removed forced edges(and their endpoints). At some cases the Δ of $\hat{G}$ drops to 0 since the vast majority of edges have been declared forced and removed along with their endpoints.

- For smaller initial matching the Δs of G and $\hat{G}$ follow mostly the same trend only a few number of different instances.

2.3 Barabási Albert

Graph characteristics Barabási Albert

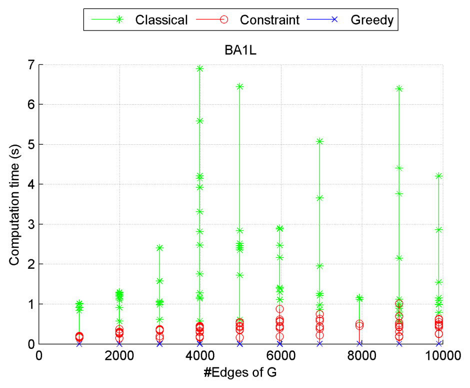
- BAxy are randomly generated graphs with a fixed number of 1000 nodes

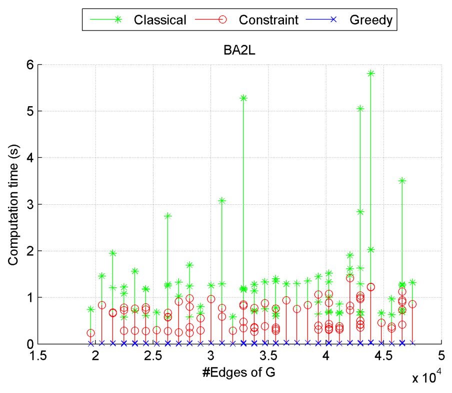
- There are 6 categories based on ini matching and # of attached edges m

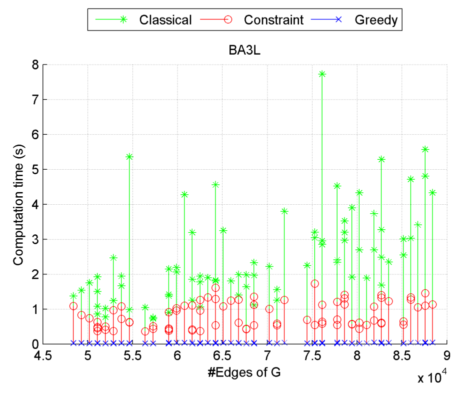
- $x = \{L, S\}$, L When the initial matching is large and S when it is small

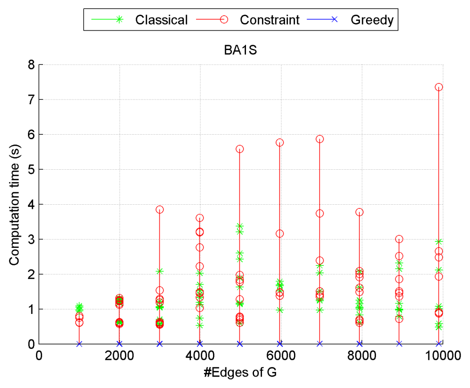
- $y = \{1, 2, 3\}$ 1 when $m = [1, 10]$, 2 when $m = [20, 50]$ and 3, $m = [50, 100]$

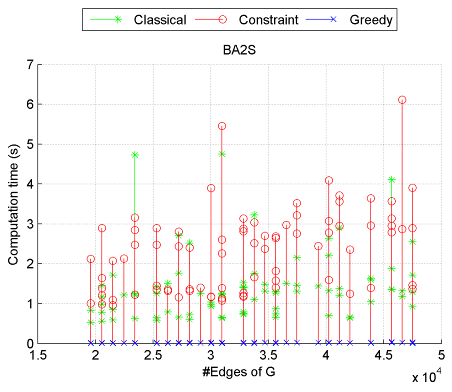
Computation time over $|E|$

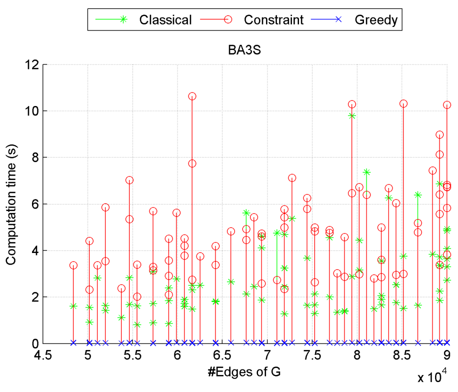








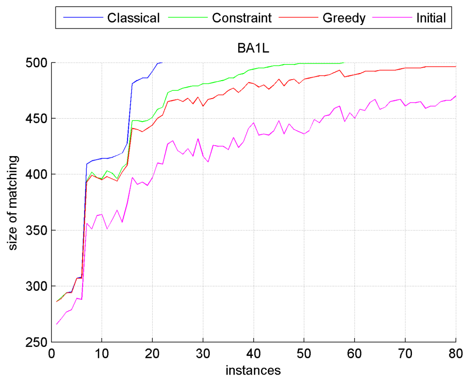


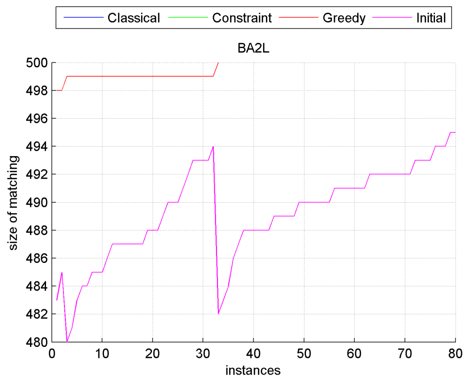


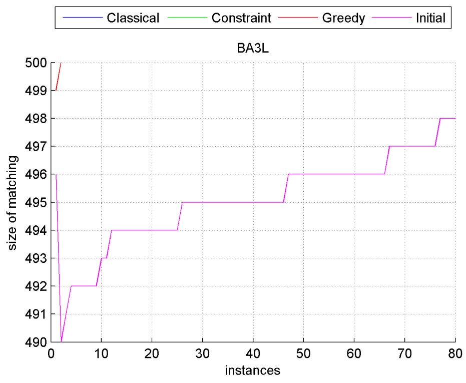
Computation time over $|E|$ - Comments

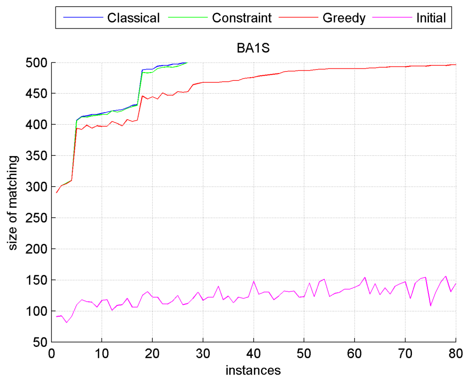
- When the initial matching is sufficiently large then the forced edges are taken out(a process which takes a fragment of the computation time) and then the classical matching algorithm (Edmonds) is applied on a smaller graph($\hat{G}$). This reduces significantly the computation time.
- When the initial matching is small then the computation time of the constraint algorithm might be larger than that of Edmonds. The reason is that the constraint algorithm need some time to compute the useless and forced edges and then Edmonds algorithm is applied on $\hat{G} \simeq G$. Thus the overall processing time is sometimes larger. .

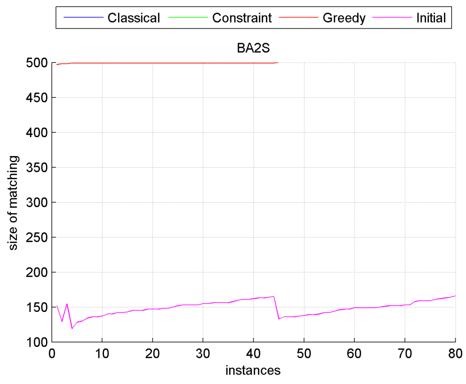
Sorted matching size

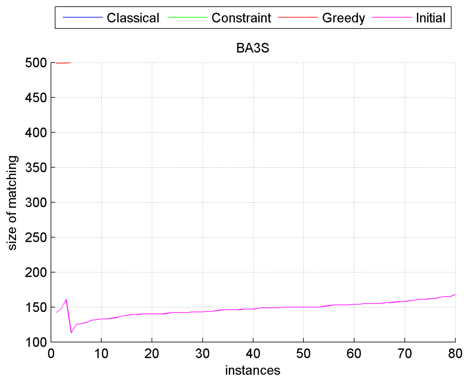












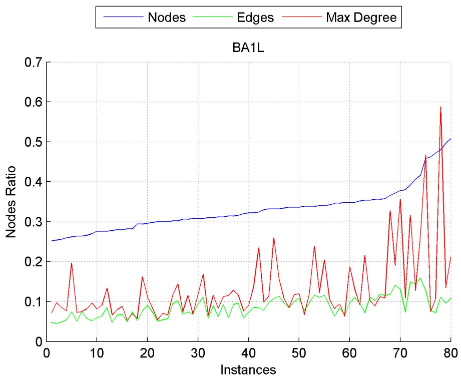
Sorted matching size-Comments

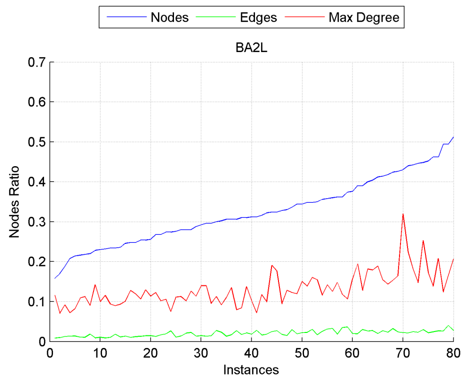
- In this graph the matching size of the classical algorithm is depicted in ascending order along with the corresponding constraint algorithm and greedy result.
- Here in the case where m is small for both cases of initial matching (small, large), the result may vary greatly starting from a value of half the perfect matching with small differences between the results of the various algorithm implementations.
- For the other cases of m the results converge.
- This is explained as for small values of m the graph is disconnected.
- In the matching size difference set of figures a very noticing difference value appears. For the cases of small m(the number of existing nodes that a new node is connected to at the creation of the graph) we can see that the difference in the

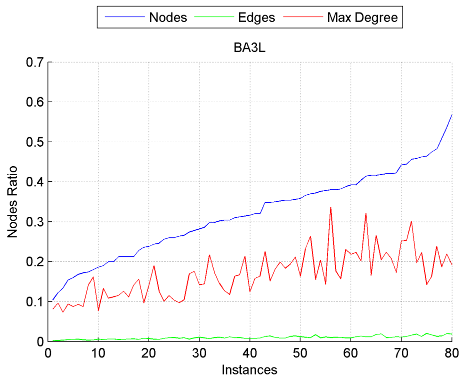
matching size compared to that of the perfect matching could be up to 40% of that of the matching size. So there are occasions where the matching size is 200 edges when the perfect matching would have been 500.

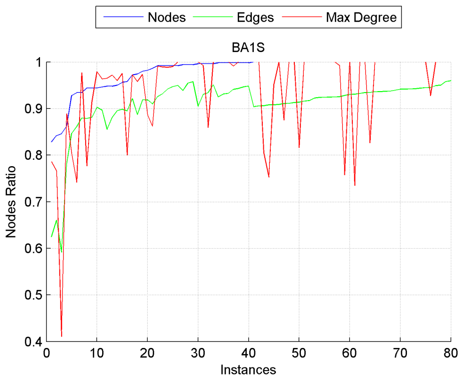
- the difference between the Constraint matching and the classical matching and the difference of the constraint matching and that of the greedy matching do not show significant values.

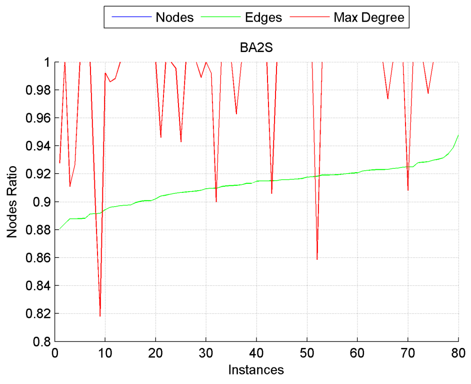
Sorted Ratios

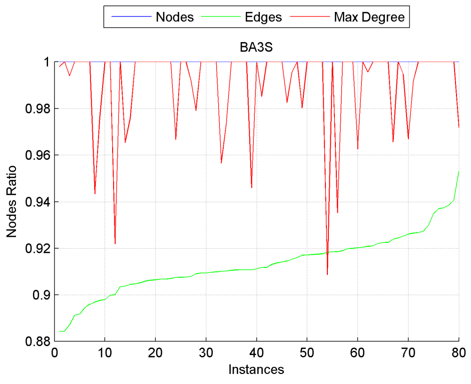










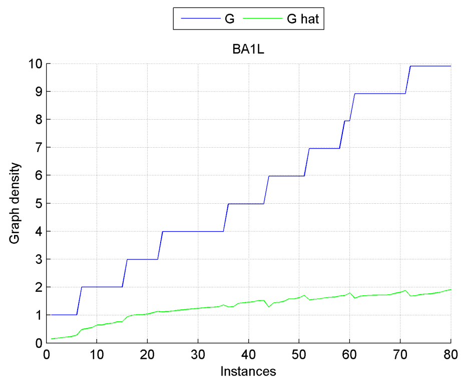


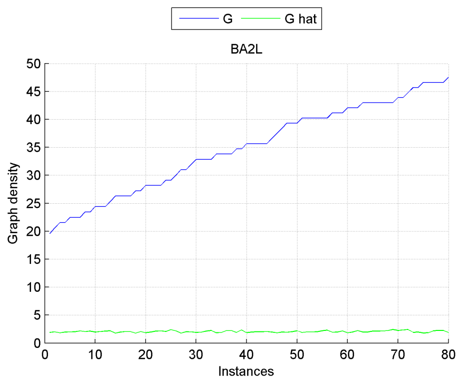
Sorted Ratio s -Comments

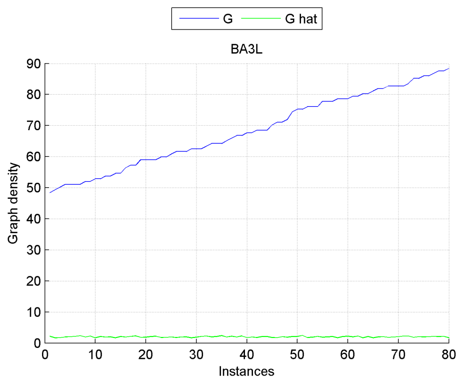
- This set of figures depicts the ratios: $\frac{\hat{V}}{V}$, $\frac{\hat{E}}{E}$, $\frac{\hat{\Delta}}{\Delta}$,in ascending order of $\frac{\hat{V}}{V}$

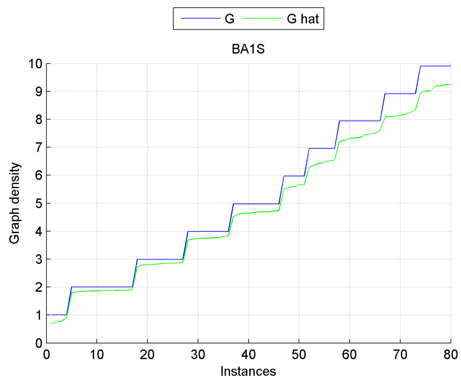
- It is evident that when the initial matching is large the amount of forced edges (and their endpoints) removed is high, thus the ratio of Nodes and edges is very low. For higher probabilities almost all edges are removed as useless.
- For a smaller initial matching almost no forced edges (and their endpoints) are removed which we conclude be the node ration being close to 1. There is some small amount of useless edges removed though since the ratio of edges is less but close to 1
- The ratio of Δ for large initial matching follows the trend of the node ratio, yet for smaller initial matching it deviates significantly.

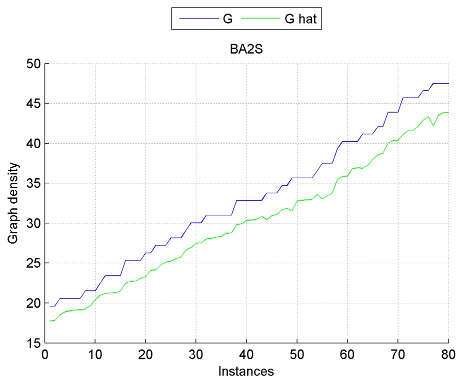
Sorted $\frac{|E|}{|V|}$

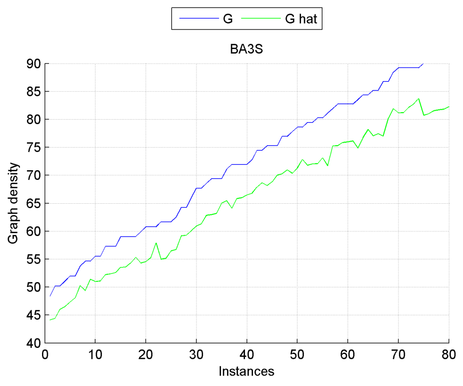












Sorted $\frac{|E|}{|V|}$ - Comments

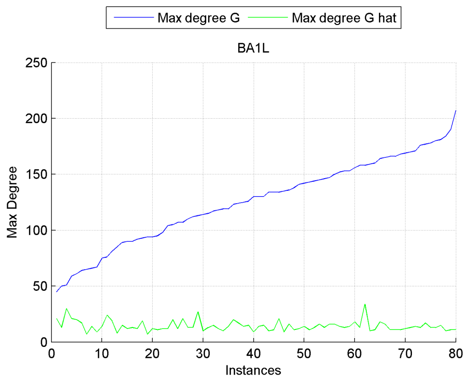
- Although as graph density the ratio $\frac{2|E|}{|V|(|V|-1)}$ is considered, in this set of figures we evaluate the correlation of the ratio $\frac{|E|}{|V|}$ of G and $\hat{G}$, since the density of $\hat{G}$ could acquire values higher than that of G

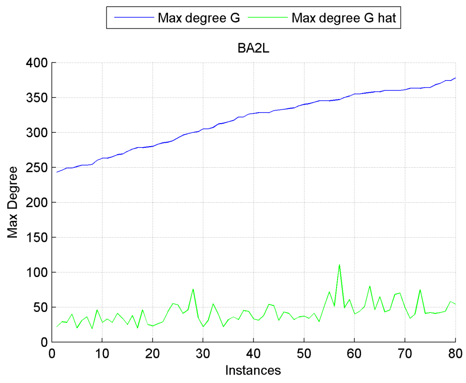
- The figures are calculated in ascending order for the ratio of G

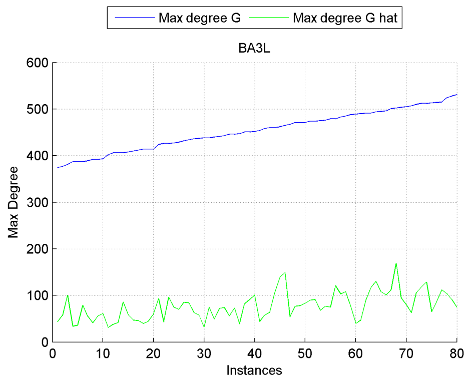
- For the case of large initial matching the two ratios for G and $\hat{G}$ remain almost constant (to the extent of the edge attraction m of each graph), since removing forced edges(and their endpoints) alters the $|E|$ and $|V|$ proportionately.

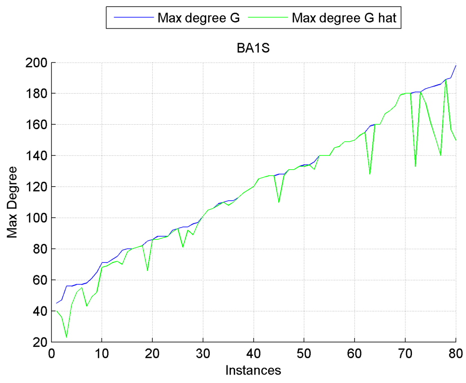
- For the case of small initial matching we observe some deviation in the values of the ration for $\hat{G}$, and that is explained with the removal of only useless edges which only alters the $|E|$ and keeps $|V|$ in the original values.

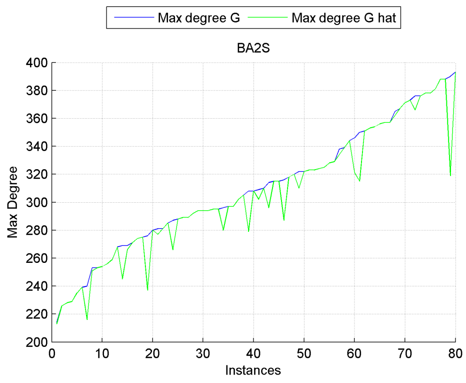
Sorted Δ

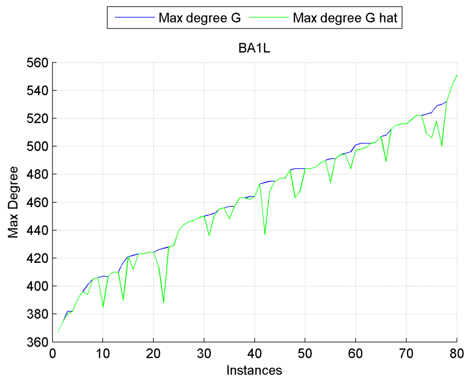












Sorted Δ-Comments

- In this set of figures the ascending order of the Δ of G is associated with the Δ of $\hat{G}$

- There is a significant jump of the Δ for each graph category which tells that that we are not comparing a uniform distribution of random graphs and based on the edge attraction m the graphs can vary in form.

- For larger initial matching we observe the difference in the Δs of G and $\hat{G}$ given the removed forced edges(and their endpoints). At some cases the Δ of $\hat{G}$ drops to 0 since the vast majority of edges have been declared forced and removed along with their endpoints. What is interesting is that the Δ of $\hat{G}$ oscilates within stable limits.

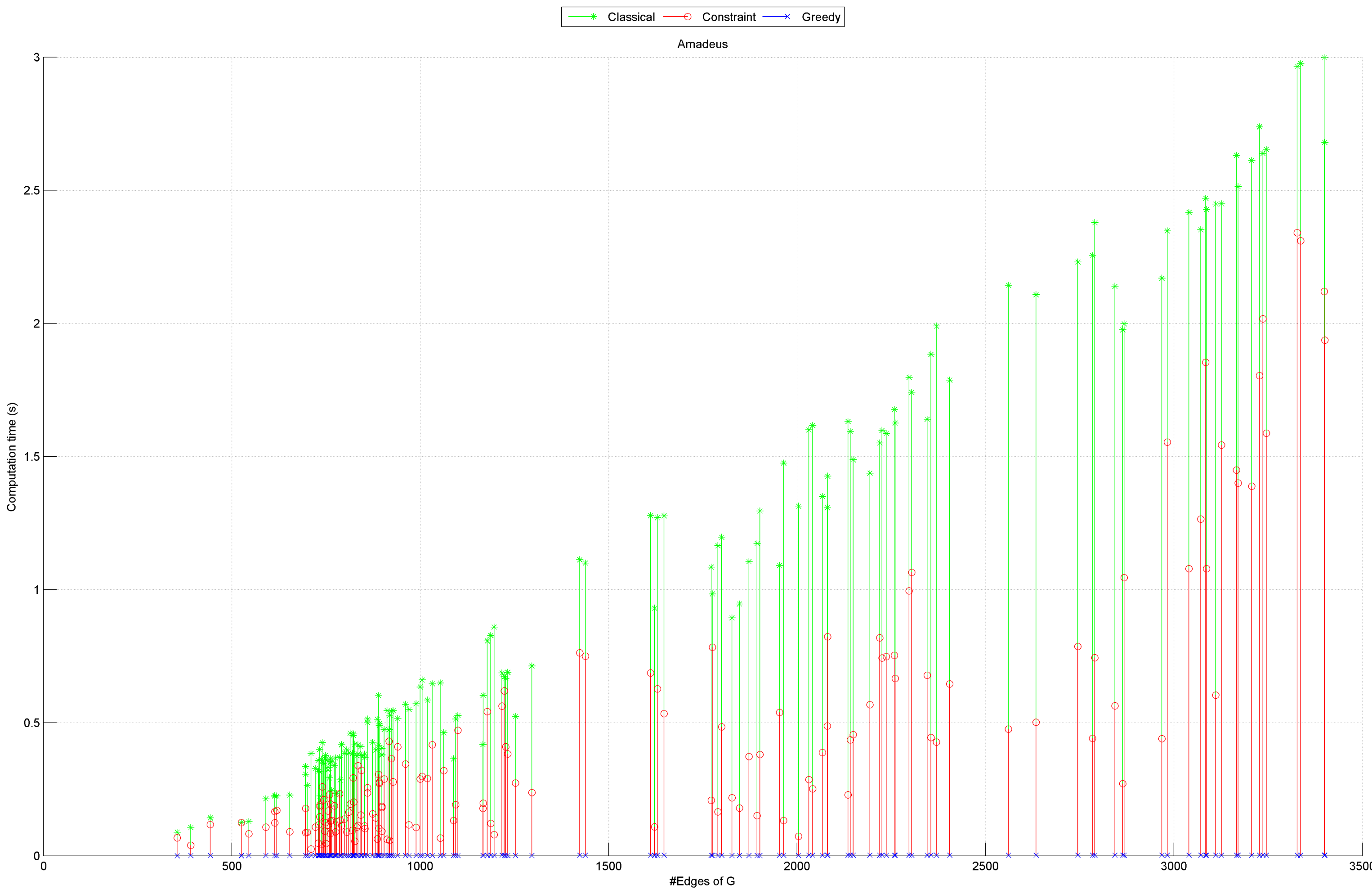
- For smaller initial matching the Δs of G and $\hat{G}$ follow mostly the same trend only a few number of different instances.

2.4 Amadeus Graphs

Graph characteristics Amadeus graphs

- These are the bipartite graphs provided by Amadeus

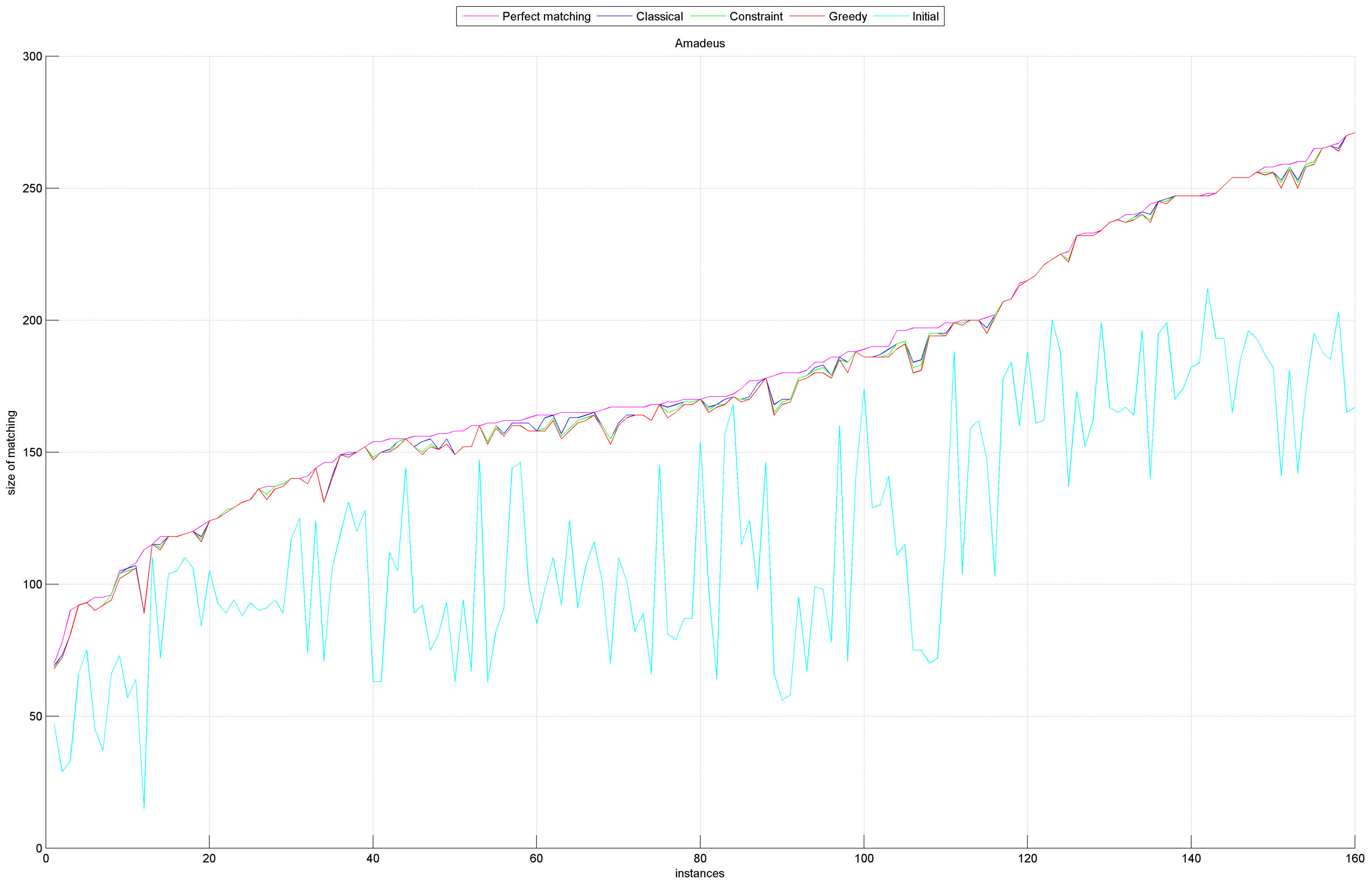
Computation time over $|E|$



Computation time over $|E|$ - Comments

- For every instance the computation time of the classical algorithm is always smaller to that of the constraint matching.
- This allows as to speculate that we always compute a smaller $\hat{G}$, which also implies that we always find a set of forced edges and useless.

Sorted matching size

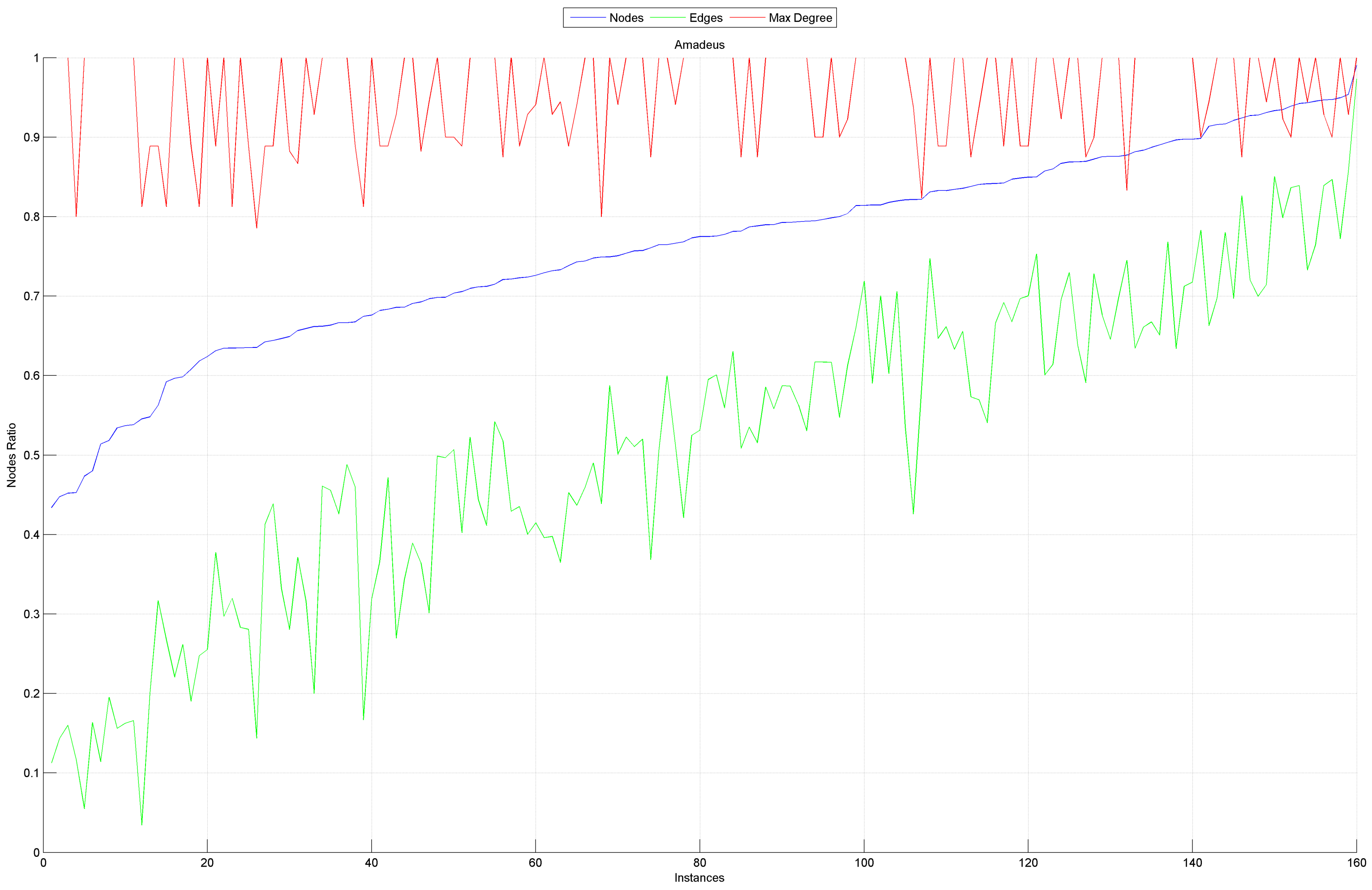


Sorted matching size - Comments

- Size of matching has an upward trend since the magnitude of the graphs is not the same for the 160 instances
- No significant deviation is noticed as per the first graph

- Except from deviation from the perfect matching (number of flights) where sometimes it could get up to $\simeq 40$
- For the Amadeus Bipartite graphs we can see that the difference of the Classical and the constraint implementation is not more than 1 edge in the matching
- There appears to be a difference of up to 24 edges between the Classical matching and the Perfect matching which is computed as the number of flights.
- The greedy algorithm performs well with only a few instances deviating slightly form the Constraint algorithm result

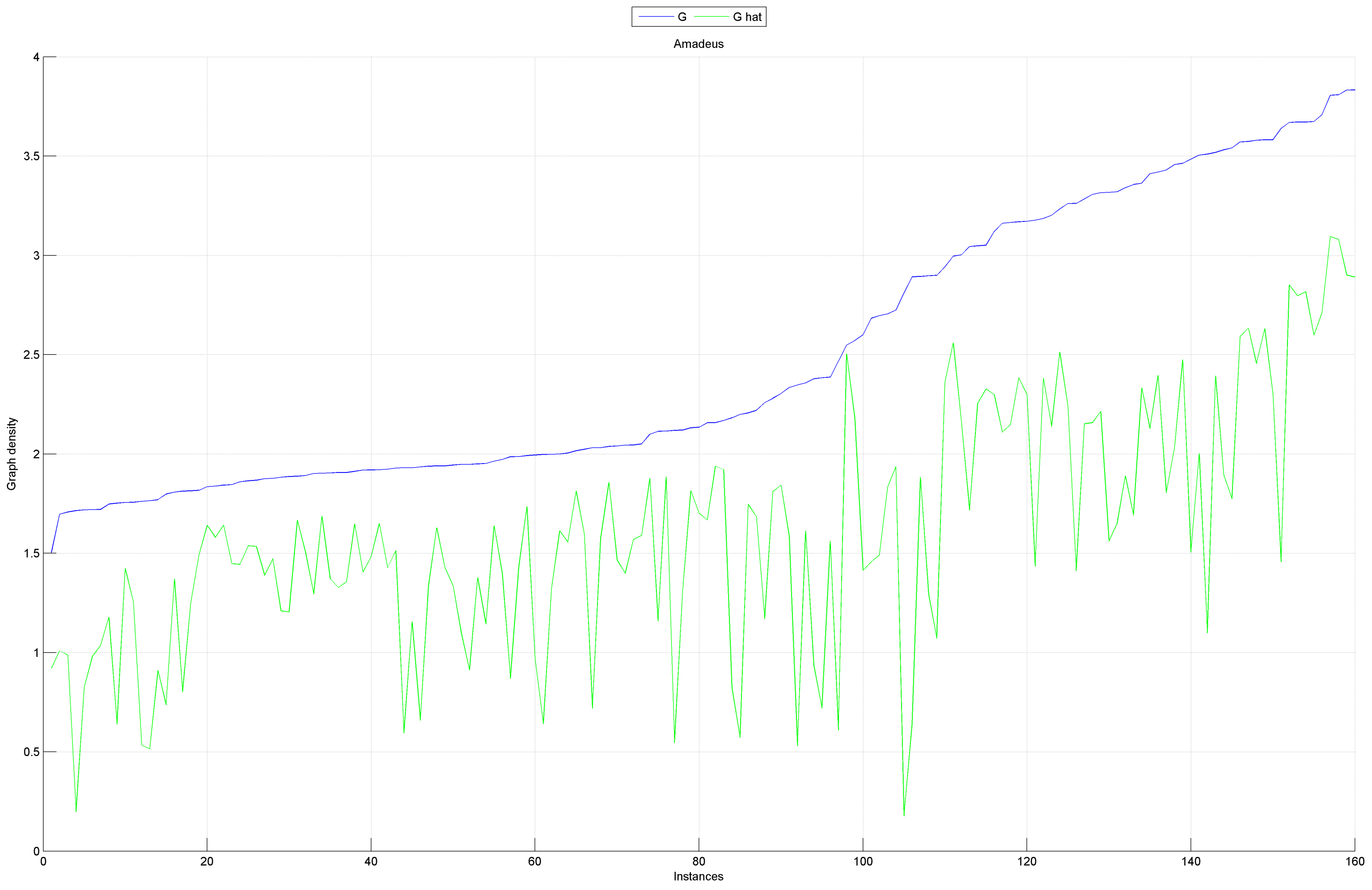
Sorted Ratios



Sorted Ratios - Comments

- ?????

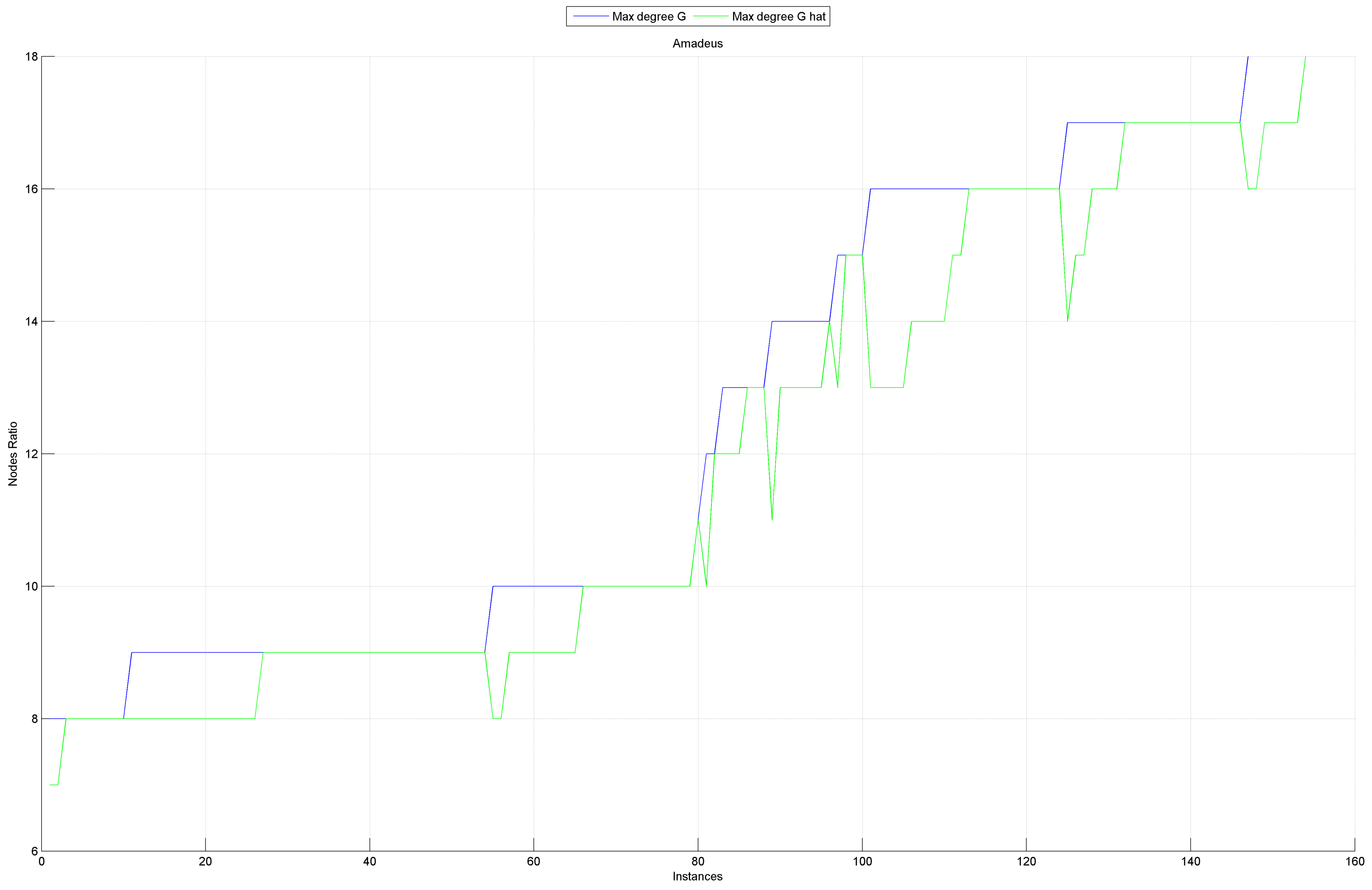
Sorted $\frac{|E|}{|V|}$



Sorted $\frac{|E|}{|V|}$ - Comments

- Density of the graphs shows that on average we have 2.5-3 available time slot per flight
- The line for the same value for $\hat{G}$ shows that there is a large deviation for every instance which allows for the hypothesis that the amount of useless+forced edges of $\hat{G}$ has no obvious correlation to that of G

Sorted Δ



Sorted Δ-Comments

- As per the previous related graphs for Δ there is in significant drop in the Δ of G and $\hat{G}$

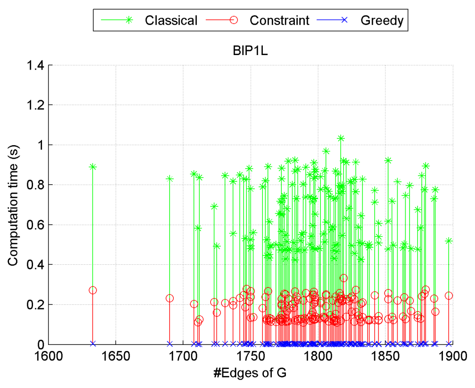
2.5 Bipartite

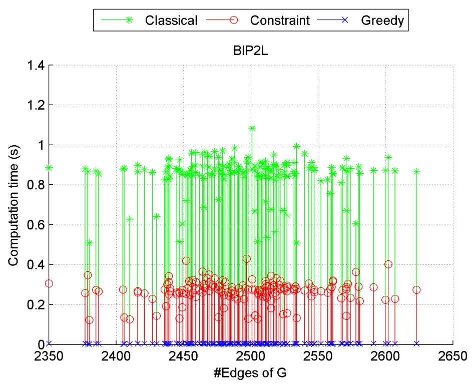
Graph characteristics Bipartite random Graphs

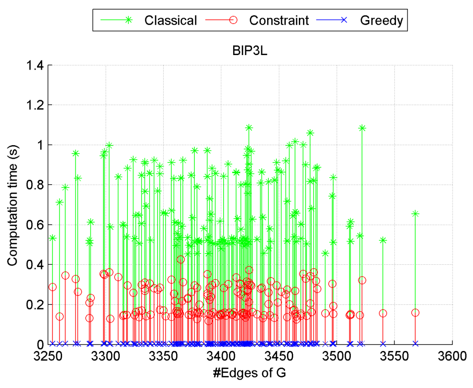
- BIPxy are randomly generated graphs with a fixed number of 1000(n=500 and m=500) nodes
- There are 6 categories based on ini matching and probability p

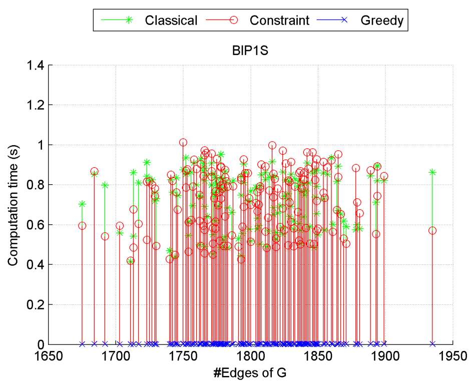
- Probability p has been chosen as such, in order to simulate the $\frac{|E|}{|V|} \simeq 2.5$ of the Amadeus bipartite graphs
- So in order to have about 2-4 edges for every node we choose accordingly the probability p to choose one of the $n * m$ possible edges.
- $x = \{L, S\}$, L When the initial matching is large and S when it is small
- $y = \{1, 2, 3\}$ 1 when $p = \frac{1.8}{250}$, 2 when $p = \frac{2.5}{250}$ and 3, when $p = \frac{3.4}{250}$

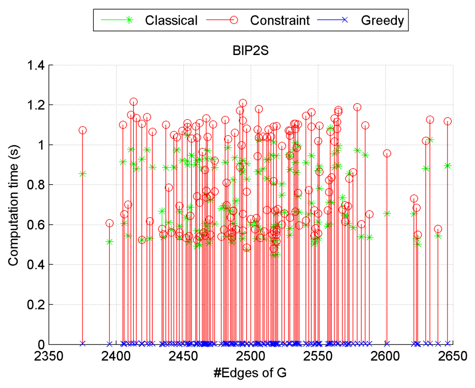
Computation time over $\#$

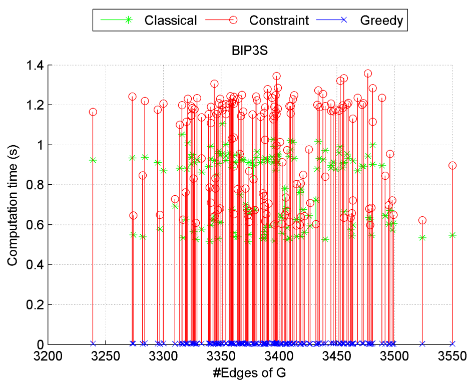








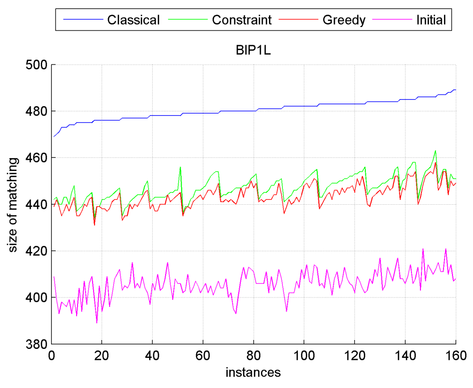


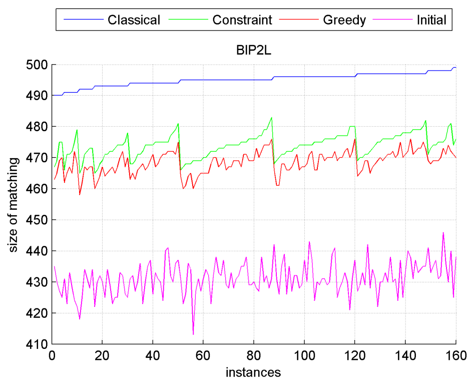


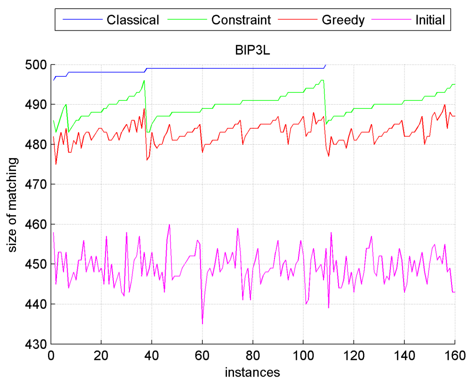
Computation time over$\#$edges-Comments

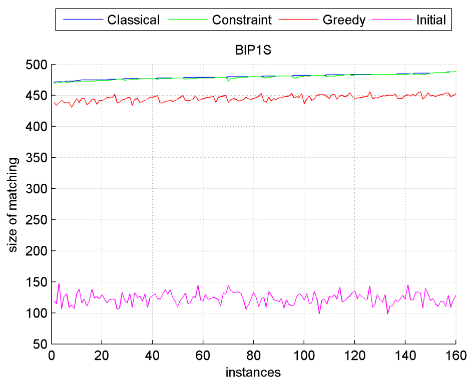
- ?????

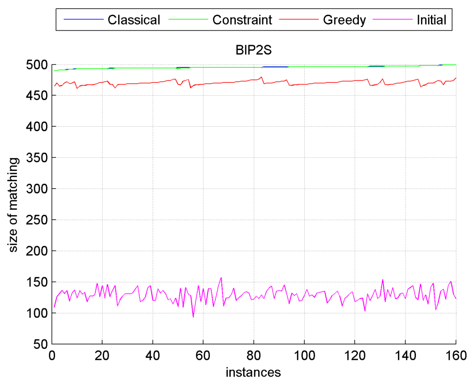
Sorted matching size

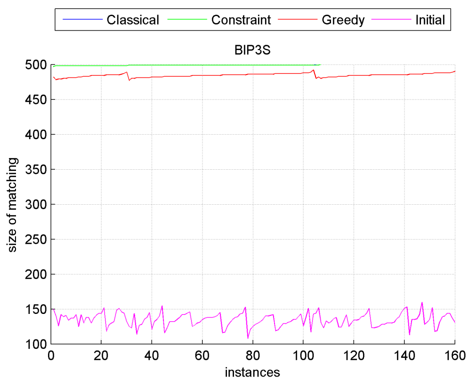










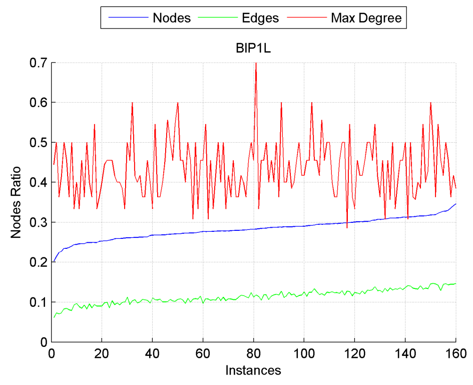


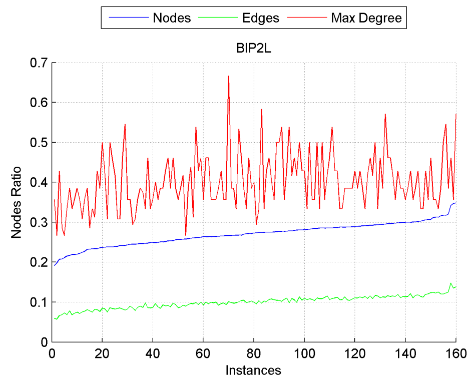
Sorted matching size-Comments

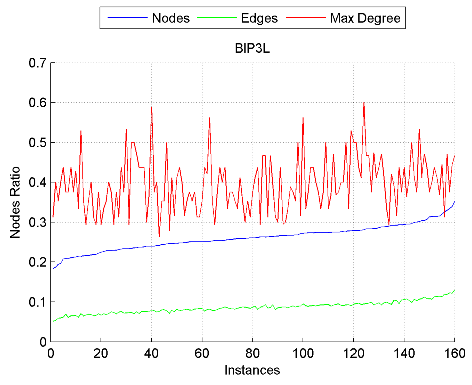
- In these set of graphs one can instantly notice that only a few instances with high probability can reach a matching size equal to the perfect matching
- For the case of the initial matching we witness that there is a gap between the line of the classical matching and those of the constraint and greedy results, suggesting smaller matching sizes of 10-40 edges.
- For the case of the small initial matching the result of the constraint matching is now close to the result of the classical matching, yet the greedy algorithm still shows a relevantly big difference.

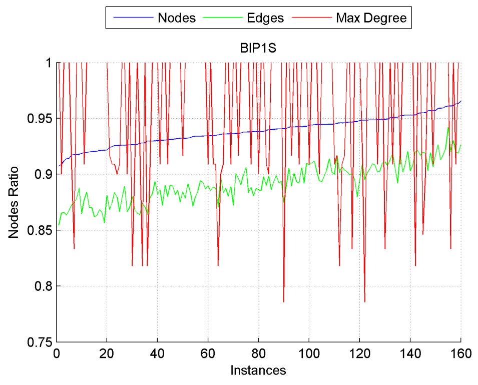
- In this set of bipartite graphs we can witness that the result of the Constraint matching shows a difference with the classical matching when the initial matching is large. The difference is smaller as the probability gets larger.
- For the case of the small initial matching the above phenomenom does not occur, yet the greedy results deviates from that of the constraint matching
- As per the pefrect matching compared to the classical matching a difference of 5-10 edges is evident at almost all cases.

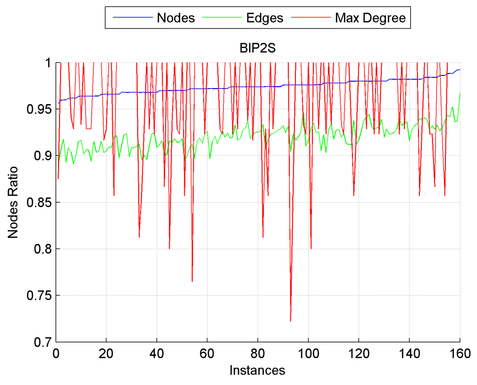
Sorted Ratios

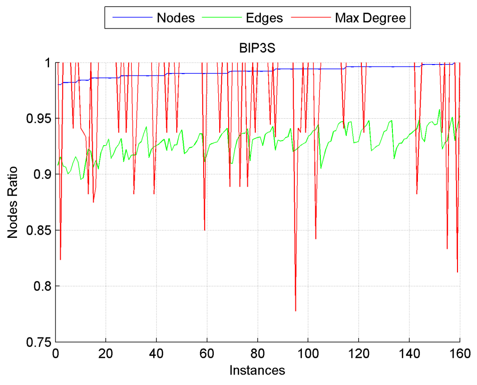










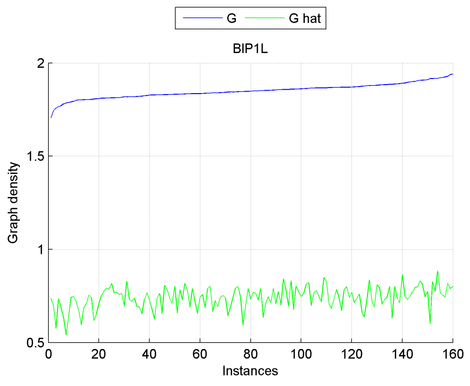


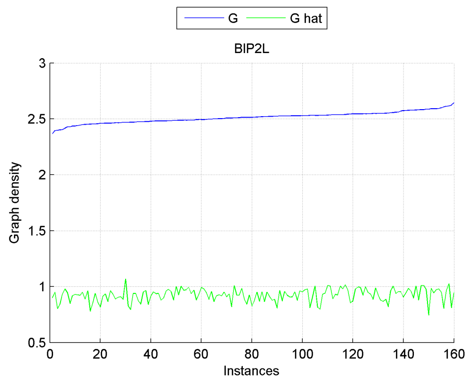
Sorted Ratios-Comments

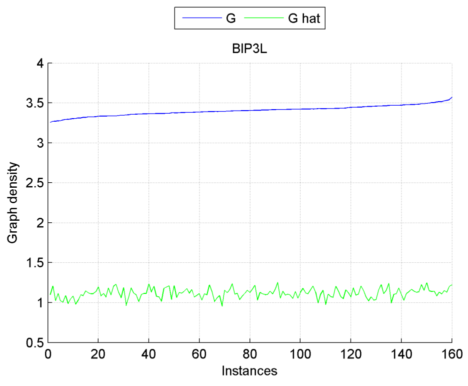
- Great max degree deviation

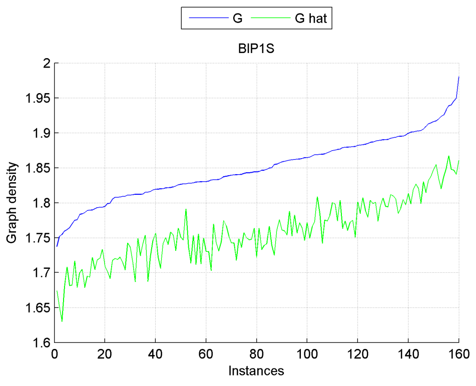
- Regardless of the probability the ration of nodes and edges seems to have similar behaviour when compared to the respective graphs of the same matching scale.
- Same gap between nodes ration and edges ration for all cases

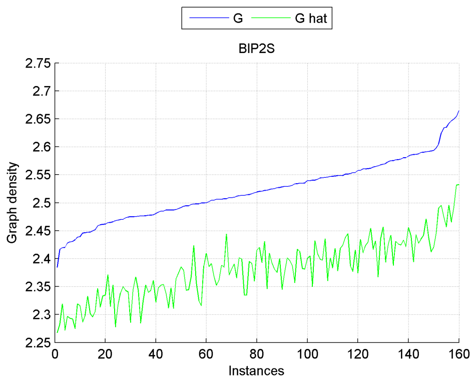
Sorted $\frac{|E|}{|V|}$

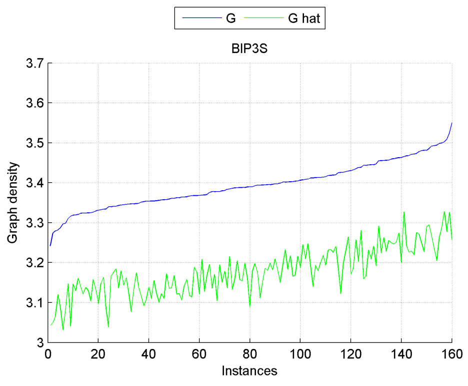








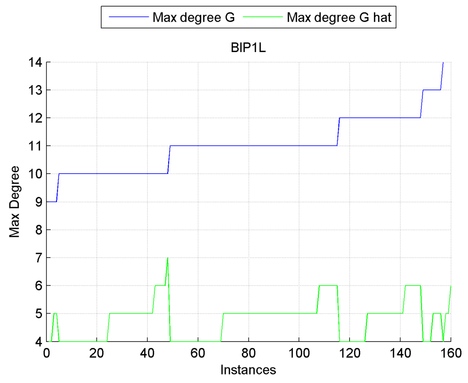


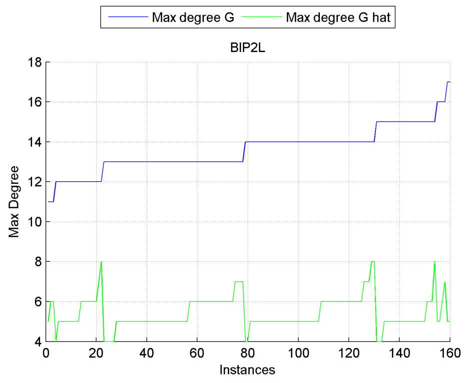


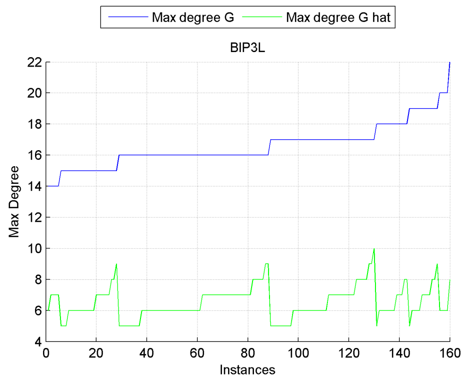
Sorted $\frac{|E|}{|V|}$ - Comments

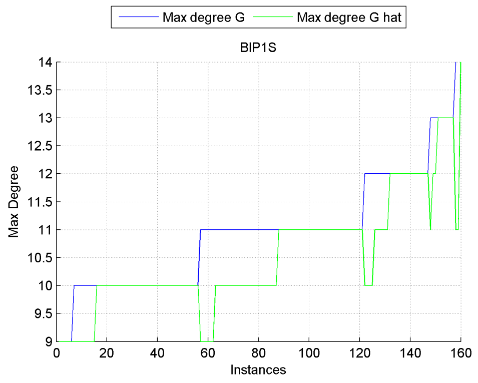
- In these set of figures we witness that the $\frac{|E|}{|V|}$ acquires values relevant to those of the amadeus graphs. The random graphs were constructed to depict this similarity.
- For smaller initial matching the ratios for G and $\hat{G}$ show the same upward trend and also they acquire closer values. It represents the decrease of the average number of forced edges in comparison to the larger matching cases

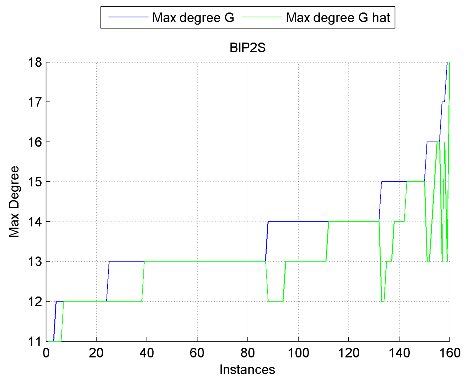
Sorted Δ

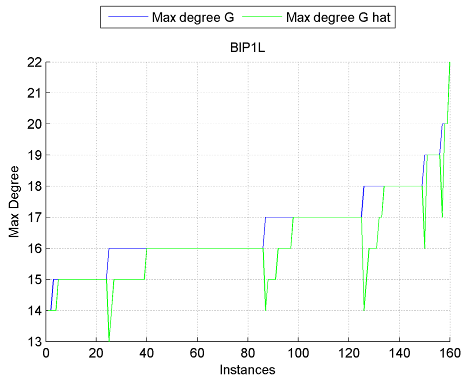












Sorted Δ-Comments

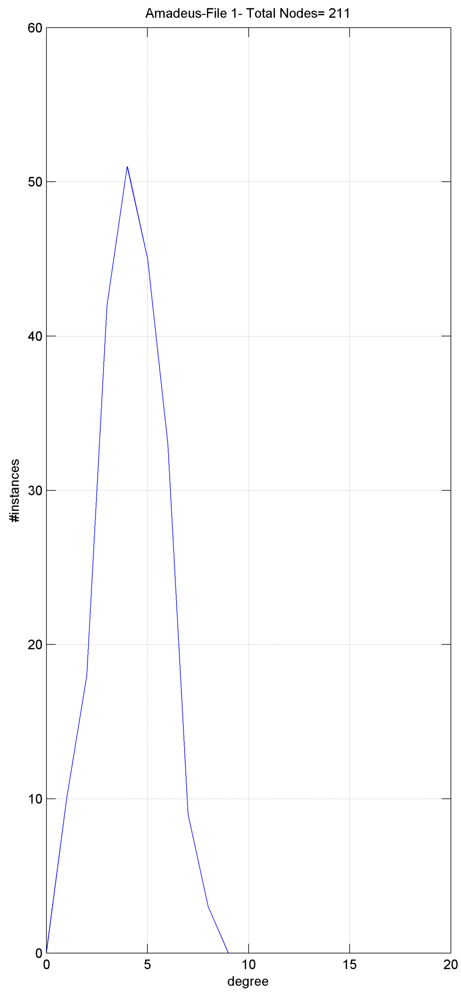
- Δ acquires the same min and max values for each probability case (expected)
- Gap in the case of large initial matching

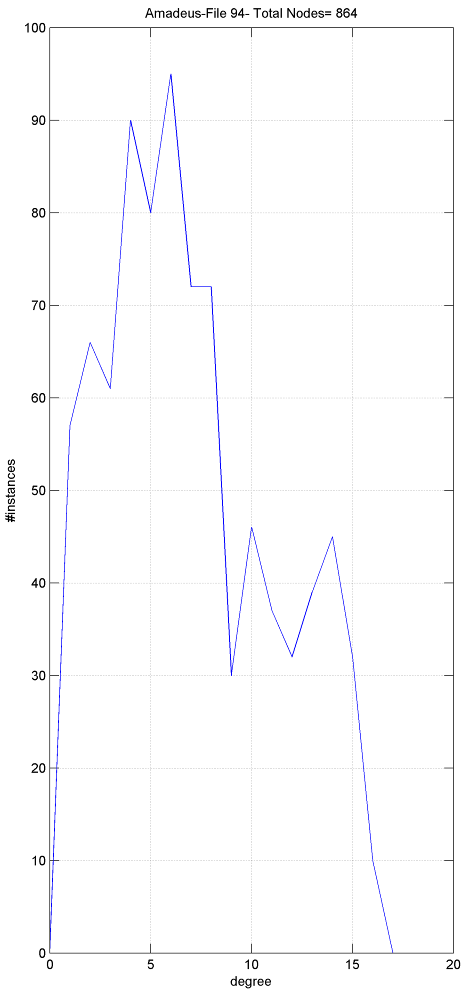
2.6 Bipartite with degree seq

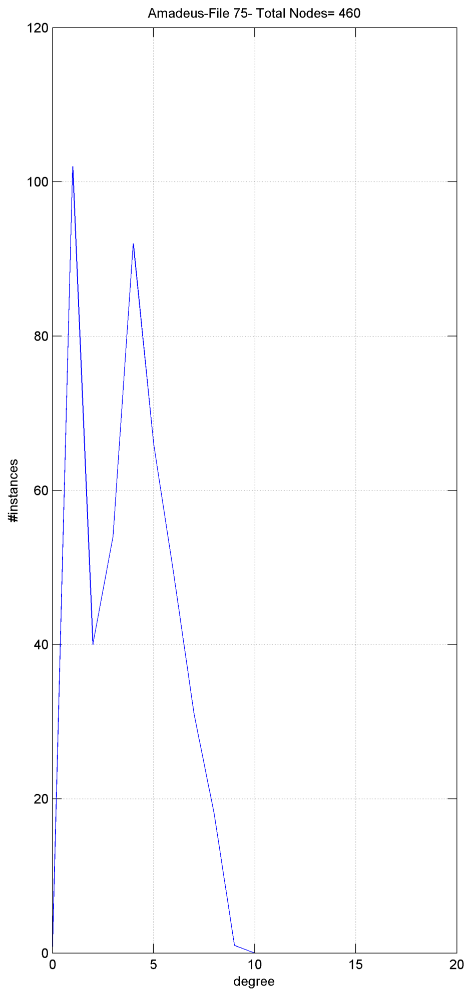
Graph characteristics

- DEQBIPxy are randomly generated graphs based on three different types of degree distribution of the Amadeus graphs

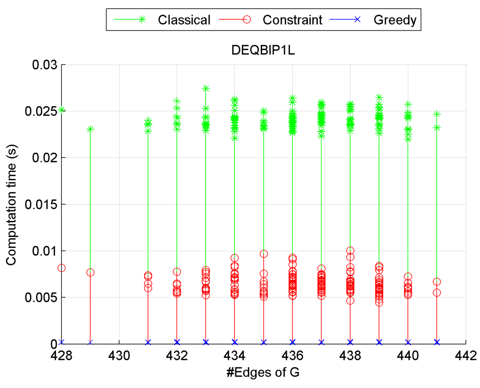
- There are 6 categories based on ini matching and degree distribution
- $x = \{L, S\}$, L When the initial matching is large and S when it is small
- $y = \{1, 2, 3\}$ 1 when degree distribution is based on file 2 when degree distribution is based on file and 3 when degree distribution is based on file

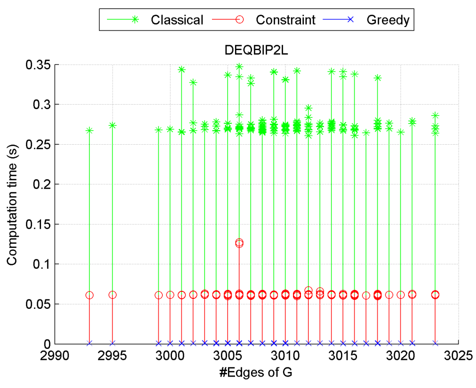


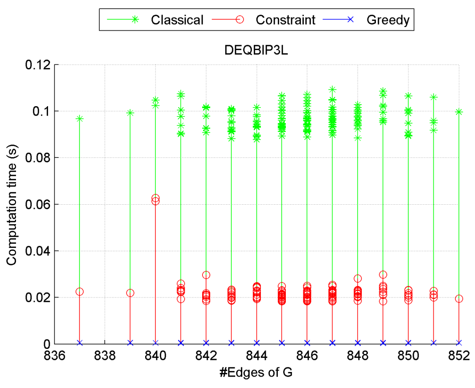


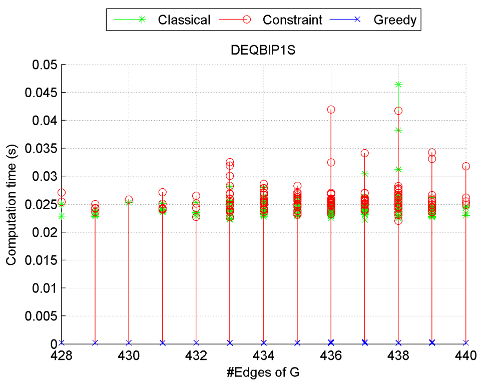


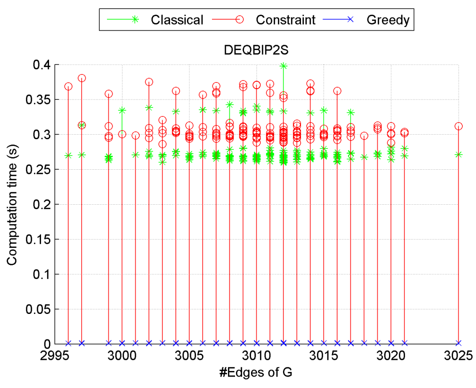
Computation time over #

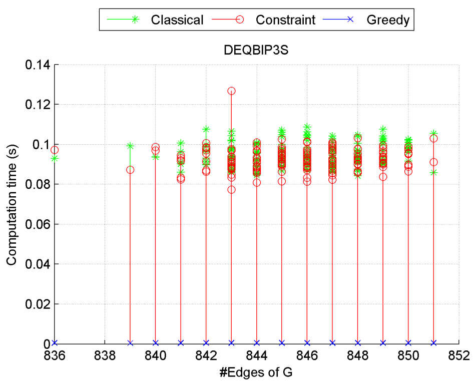








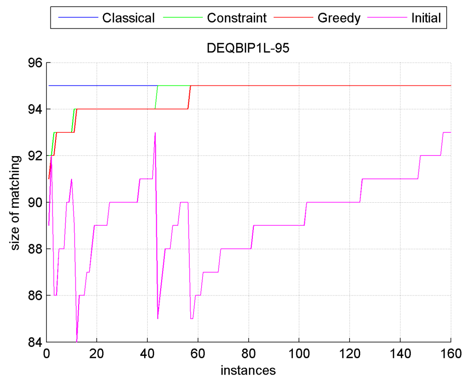


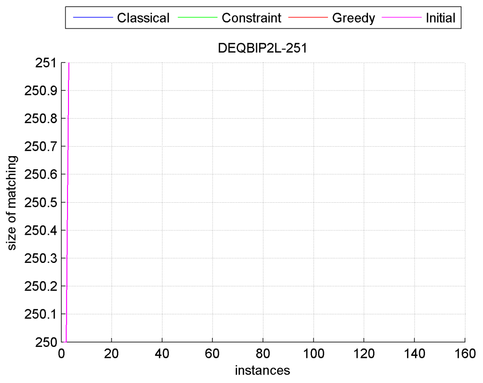


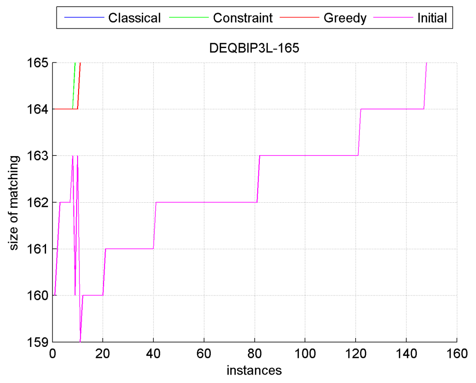
Computation time over$\#$edges-Comments

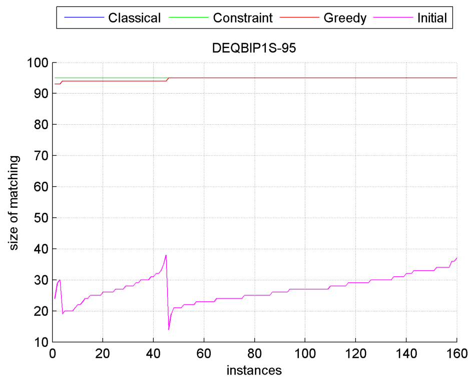
- Greedy algorithm takes minimum amount of time
- The smaller the initial matching the more smaller the difference in comoutation time of Constraint and classical algorithm
- Due to preprocessing and the fact that G and $\hat{G}$ are the almost identical , the total time needed to apply the constraint algorithm is in some cases bigger.

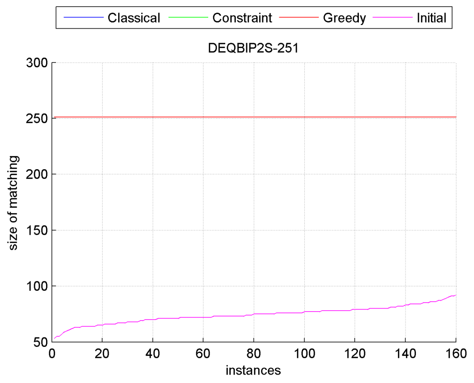
Sorted matching size

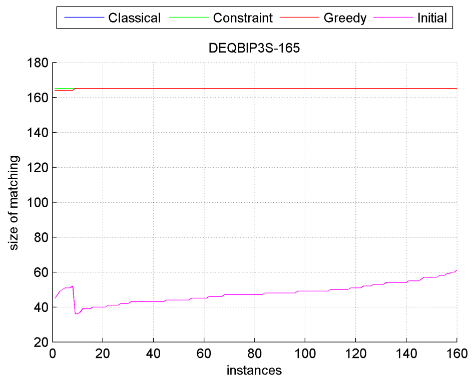








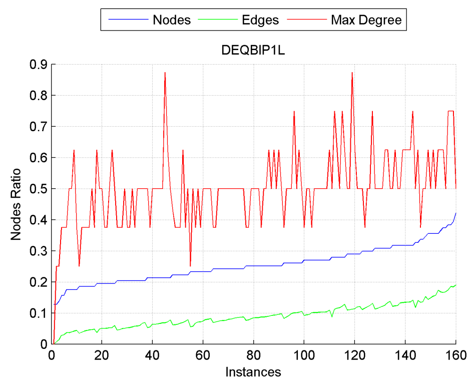


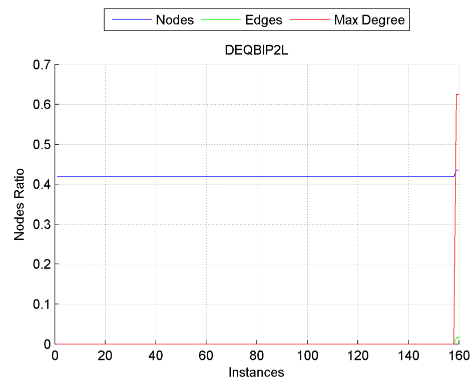


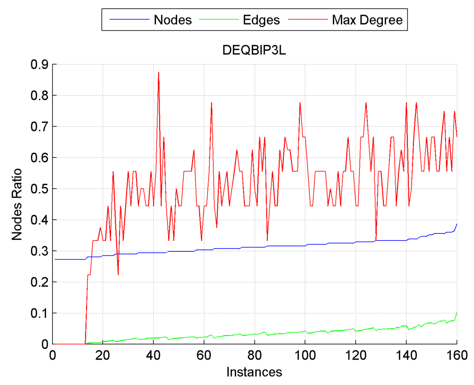
Sorted matching size-Comments

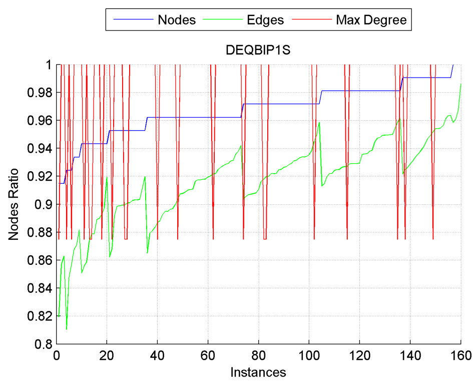
- Classical matching is equal to perfect matching in every case
- Constraint and greedy have identical behaviour
- The larger the spectrum of the degree sequence the more efficient the matching

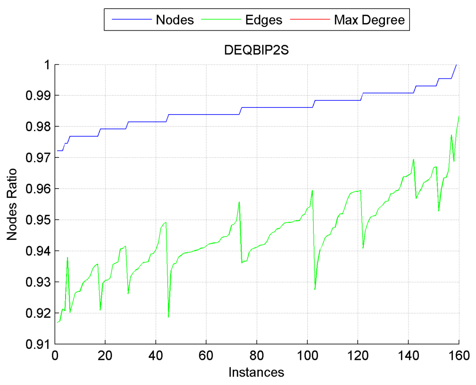
Sorted Ratios

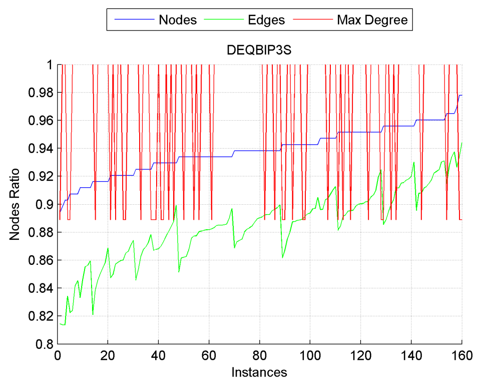










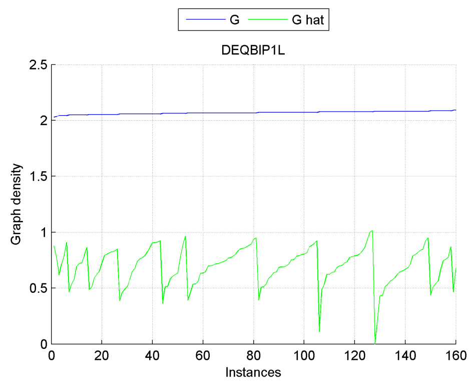


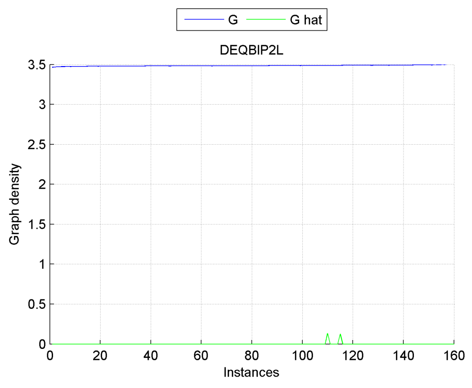
Sorted Ratios-Comments

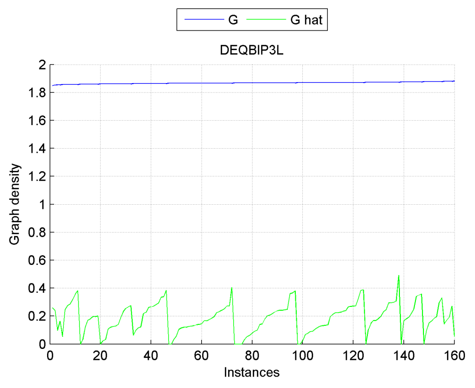
- With large initial matching most of the edges are considenred uselles or forced and are removed thus the ratio is close(if not equal to) zero

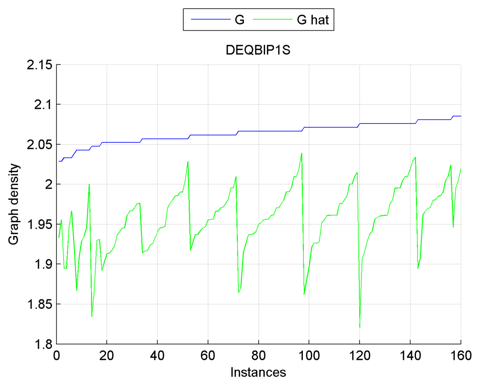
- For small matching and large degree sequence sectrum hte max degree remains the same. Only a few forced edges (and their nodes) are removed apparently no nodes with max degree

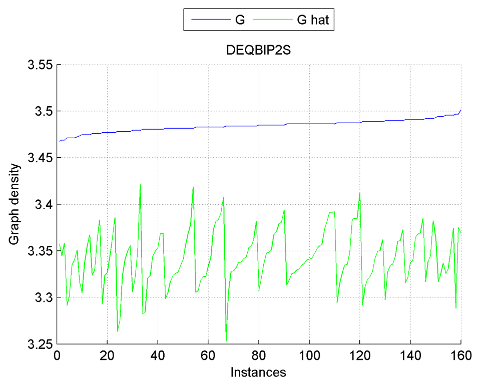
Sorted $\frac{|E|}{|V|}$

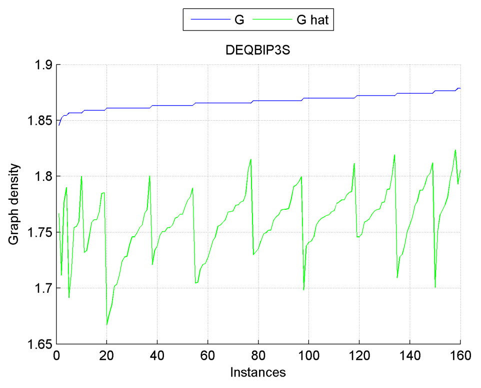










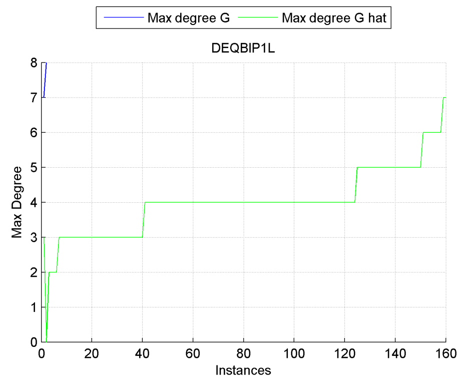


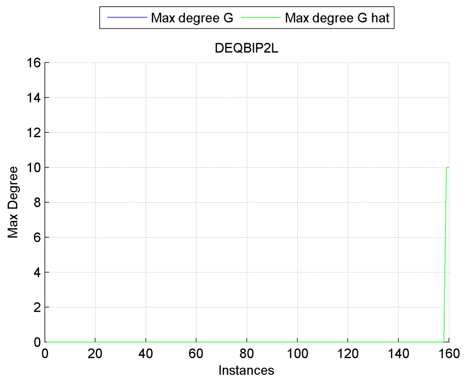
Sorted $\frac{|E|}{|V|}$ - Comments

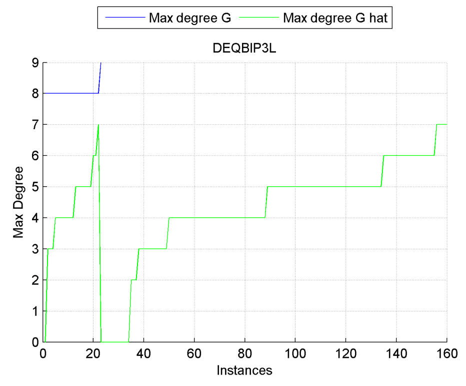
- In these set of figures we witness that the $\frac{|E|}{|V|}$ acquires values relevant to those of the amadeus graphs. The random graphs were constructed to depict this similarity.

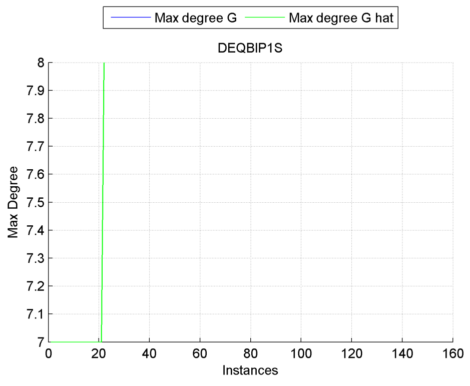
- For smaller initial matching the ratios for G and $\hat{G}$ show the same upward trend and also they acquire closer values. It repsresent the decrease of the average number of forced edges in comparison to the larger matching cases

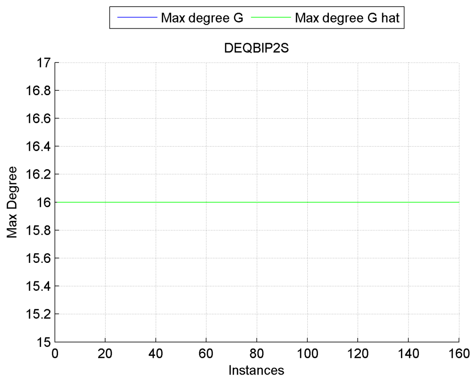
Sorted Δ

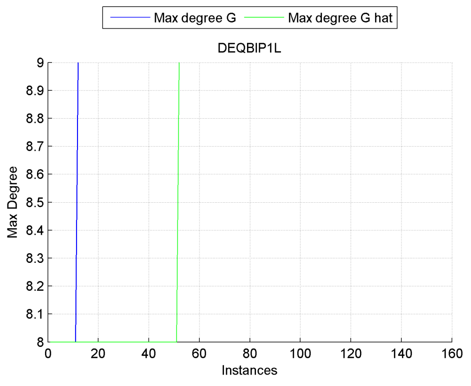












Sorted Δ-Comments

- Big difference in max degree in large initial matching
- Almost identical behaviour in the case of small initial matching

Appendix A

BipartiteGraphGenerator.py

```
from __future__ import division
import networkx as nx

from networkx.algorithms import approximation
import collections
import random
import math
from networkx.algorithms import bipartite
if __name__ == '__main__':

    for i in range(1,481):#waqs 481
        #index=1
        #n=random.randint(100, 1000)

        n1=95 #was 500
        me1=118

        n2=251 #was 500
        me2=903

        n3=165 #was 500
        me3=412

        m=0
        m1=0
        #p=random.random()
        p1=1.8/250
        p2=2.5/250
        p3=3.4/250
        print i
        #p=0.5
        t1=random.randint(0, 1)
        #print i
        #m2=random.randint(n/2-1, n/2+1)
        G = nx.Graph()
        ma = nx.Graph()

        aseq1=[1, 2, 3, ..., 4, 5, 3]


        bseq1=[4, 3, 3,..., 1, 2, 2, 1]

        aseq2=[16, 14, 14,..., 14, 12, 11]
        bseq2=[0, 0, 0,..., 0, 0, 0]

        aseq3=[8, 7, 8,..., 8, 8, 8]
        bseq3=[0, 0, 0,..., 0, 0, 0]
```

```
#print G.number_of_edges()
if i<=160:
    G=bipartite.configuration_model(aseq1, bseq1, create_using=None, seed=None
    A=nx.adjacency_matrix(G)
    for ii in range(0,n1+me1):
        ma.add_node(ii)
    A=[R.todense().tolist()[0] for R in A]
#print"num",ma.number_of_nodes()
    rl=random.sample(range(0,n1+me1), n1+me1)
    m=0
    done_vertices = set()
    for p in rl:
    #print"in for"
    #done_vertices.add(p)
        neighbors = []#list()
        for iii in range(0,n1+me1):
            if A[p][iii]==1:
                neighbors.append(iii)
        random.shuffle(neighbors)
        continue_loop = True
        degree = len(neighbors)
    #print degree
        j = 0
        p4=random.random()
        while continue_loop and (j<degree):
        #print "in loop"
            next = neighbors[j]
            if next not in done_vertices and p not in done_vertices:
                done_vertices.add(p)
                done_vertices.add(next)
                if p4>=0.7:
                #print "added edge"
                    ma.add_edge(p,next)
                    m=m+1
                continue_loop = False
            j += 1

#print matching_listOfedge
#print ma.edge
    MA=nx.adjacency_matrix(ma)
#print MA
    MA=[R.todense().tolist()[0] for R in MA]
#print MA
    B=[" ".join(map(str,R)) for R in A]
    MB=[" ".join(map(str,R)) for R in MA]

    C="\n".join(B)+"\n"
    MC="\n".join(MB)+"\n"

#print n
#print m
#print m1
    filea=str(i)+".txt"
    with open(filea,'w') as f:
        f.write(str(n1+me1))
        f.write("\n")
        f.write(str(n1+me1))
        f.write("\n")
        f.write(str(m))
        f.write("\n")
        f.write("\n")
        f.write(C)
        f.write("\n")
        f.write(MC)
elif i>160 and i<321:
    G=bipartite.configuration_model(aseq2, bseq2, create_using=None, seed=Non
    A=nx.adjacency_matrix(G)
    for ii in range(0,n2+me2):
        ma.add_node(ii)
    A=[R.todense().tolist()[0] for R in A]
#print"num",ma.number_of_nodes()
    rl=random.sample(range(0,n2+me2), n2+me2)
    m=0
    done_vertices = set()
    for p in rl:
    #print"in for"
    #done_vertices.add(p)
        neighbors = []#list()
        for iii in range(0,n2+me2):
            if A[p][iii]==1:
                neighbors.append(iii)
        random.shuffle(neighbors)
        continue_loop = True
        degree = len(neighbors)
    #print degree
        j = 0
        p4=random.random()
        while continue_loop and (j<degree):
            #print "in loop"
            next = neighbors[j]
            if next not in done_vertices and p not in done_vertices:
                done_vertices.add(p)
                done_vertices.add(next)
                if p4>=0.7:
                    #print "added edge"
                    ma.add_edge(p,next)
                    m=m+1
                continue_loop = False
            j += 1

#print matching_listOfedge
#print ma.edge
    MA=nx.adjacency_matrix(ma)
#print MA
    MA=[R.todense().tolist()[0] for R in MA]
#print MA
    B=[" ".join(map(str,R)) for R in A]
    MB=[" ".join(map(str,R)) for R in MA]
```

```
            C="\n".join(B)+"\n"
            MC="\n".join(MB)+"\n"

        #print n
        #print m
        #print m1
            filea=str(i)+".txt"
            with open(filea,'w') as f:
                f.write(str(n2+me2))
                f.write("\n")
                f.write(str(n2+me2))
                f.write("\n")
                f.write(str(m))
                f.write("\n")
                f.write("\n")
                f.write(C)
                f.write("\n")
                f.write(MC)
        else:
            G=bipartite.configuration_model(aseq3, bseq3, create_using=None, seed=None
            A=nx.adjacency_matrix(G)
            for ii in range(0,n3+me3):
                ma.add_node(ii)
            A=[R.todense().tolist()[0] for R in A]
        #print"num",ma.number_of_nodes()
            rl=random.sample(range(0,n3+me3), n3+me3)
            m=0
            done_vertices = set()
            for p in rl:
            #print"in for"
            #done_vertices.add(p)
                neighbors = []#list()
                for iii in range(0,n3+me3):
                    if A[p][iii]==1:
                        neighbors.append(iii)
                random.shuffle(neighbors)
                continue_loop = True
                degree = len(neighbors)
            #print degree
                j = 0
                p4=random.random()
                while continue_loop and (j<degree):
                    #print "in loop"
                    next = neighbors[j]
                    if next not in done_vertices and p not in done_vertices:
                        done_vertices.add(p)
                        done_vertices.add(next)
                        if p4>=0.7:
                        #print "added edge"
                            ma.add_edge(p,next)
                            m=m+1
                        continue_loop = False
                    j += 1
```

```
    #print matching_listOfedge
    #print ma.edge
        MA=nx.adjacency_matrix(ma)
    #print MA
        MA=[R.todense().tolist()[0] for R in MA]
    #print MA
        B=[" ".join(map(str,R)) for R in A]
        MB=[" ".join(map(str,R)) for R in MA]

        C="\n".join(B)+"\n"
        MC="\n".join(MB)+"\n"

    #print n
    #print m
    #print m1
        filea=str(i)+".txt"
        with open(filea,'w') as f:
            f.write(str(n3+me3))
            f.write("\n")
            f.write(str(n3+me3))
            f.write("\n")
            f.write(str(m))
            f.write("\n")
            f.write("\n")
            f.write(C)
            f.write("\n")
            f.write(MC)


print "\nRandom graphs with random matching have been generated"
```

RandomGraphGenerator.py

```
from __future__ import division
import networkx as nx
from networkx.algorithms import approximation
import collections
import random
import math
if __name__ == '__main__':

    for i in range(1,481):
        #index=1
        #n=random.randint(100, 1000)

        n=1000
        m=0
        m1=0
        #p=random.random()
        p1=math.log(n,2)/n
        p2=1/3
        p3=math.log(n,2)/n+0.1
        #p=0.5
        t1=random.randint(0, 1)
        print i
        #m2=random.randint(n/2-1, n/2+1)
        G = nx.Graph()
        ma = nx.Graph()
        #print n
        me1=random.randint(1, 10)
        me2=random.randint(20,50)
        me3=random.randint(50,100)
        if i<241:
            if i<=80:
                G=nx.fast_gnp_random_graph(n, p1, seed=None, directed=False)
            elif i>80 and i<161:
                G=nx.fast_gnp_random_graph(n, p2, seed=None, directed=False)
            else:
                G=nx.fast_gnp_random_graph(n, p3, seed=None, directed=False)
        else:
            if i<=320:
                G=nx.barabasi_albert_graph(n, me1, seed=None)
            elif i>320 and i<401:
                G=nx.barabasi_albert_graph(n, me2, seed=None)
            else:
                G=nx.barabasi_albert_graph(n, me3, seed=None)
        A=nx.adjacency_matrix(G)
        for ii in range(0,n):
            ma.add_node(ii)
        A=[R.todense().tolist()[0] for R in A]

        rl=random.sample(range(0,n), n)
        m=0
        done_vertices = set()
        for p in rl:
            #done_vertices.add(p)
            neighbors = []#list()
            for iii in range(0,n):
                if A[p][iii]==1:
                    neighbors.append(iii)
            random.shuffle(neighbors)
            continue_loop = True
            degree = len(neighbors)
            j = 0
            p2=random.random()
            while continue_loop and (j<degree):
                next = neighbors[j]
                if next not in done_vertices and p not in done_vertices:
                    done_vertices.add(p)
                    done_vertices.add(next)
                    if p2>=0.0:     #########Larger probability for smaller initial ma
                        ma.add_edge(p,next)
                        m=m+1
                    continue_loop = False
                j += 1

        #print matching_listOfedge
        #print ma.edge
        MA=nx.adjacency_matrix(ma)
        MA=[R.todense().tolist()[0] for R in MA]
        #print MA
        B=[" ".join(map(str,R)) for R in A]
        MB=[" ".join(map(str,R)) for R in MA]

        C="\n".join(B)+"\n"
        MC="\n".join(MB)+"\n"
        #print MC
        #print n
        #print m
        #print m1
        filea=str(i)+".txt"
        with open(filea,'w') as f:
            f.write(str(n))
            f.write("\n")
            f.write(str(n))
            f.write("\n")
            f.write(str(m))
            f.write("\n")
            f.write("\n")
            f.write(C)
            f.write("\n")
            f.write(MC)

    print "\nRandom graphs with random matching have been generated"
```

kequals3MaximumMatching.py

```
from __future__ import division
import networkx as nx
from networkx.algorithms import bipartite
import matplotlib.pyplot as plt
import collections
import timeit
import xlwt
import xlrd
import math
import sys

####Case for  General Graphs
def forced_edges(G,adj_dict2,matching_dict1,num_nodes,forced):

    flagg=0
    for n in range(0,num_nodes):
        for k in adj_dict2[n]:
                if ([k]==matching_dict1[n]):
                    count=0
                    for l in range(0,num_nodes):
                            count=count+adj_dict2[l].count(k)
                    if (len(adj_dict2[n])==1) or (count==1):
                        forced.add_edge(n,k)
                        G.remove_edge(n,k)
                        G.remove_node(n)
                        G.remove_node(k)
                        #print"forced edge",n,k
                        temp=k
                        flagg=1
                        for m in range (len(adj_dict2[n])-1,-1,-1):
                            adj_dict2[adj_dict2[n][m]].remove(n)
                            adj_dict2[n].remove(adj_dict2[n][m])
                        if adj_dict2[temp]:
                            for l in range(len(adj_dict2[temp])-1,-1,-1):
                                if (adj_dict2[temp][l]!=n):
                                    adj_dict2[adj_dict2[temp][l]].remove(temp)
                                    adj_dict2[temp].remove(adj_dict2[temp][l])


                        break



def useless_edges(num_nodes,adj_dict1,matching_dict1,useless,adj_dict2,G):
    for n in range(0,num_nodes):
            for k in adj_dict1[n]:
                if ([k]!=matching_dict1[n]) and matching_dict1[n] and matching_dict1[k]
                    useless.add_edge(n,k)
                    G.remove_edge(n, k)
                    tempa=k
                    adj_dict2[n].remove(k)
                    adj_dict2[tempa].remove(n)
                    adj_dict1[tempa].remove(n)
    return G



def Greedy(num_nodes,adj_dict3,matching_dict2):

    for n in range(0,num_nodes):
        if matching_dict2[n]:
            flag2=-1
            flag3=-1
            for v in adj_dict3[n]:
                if not matching_dict2[v]:
                    flag2=v
                    break
            for p in adj_dict3[matching_dict2[n][0]]:
                if not matching_dict2[p]:
                    flag3=p
                    break

            if (flag2!=-1) and (flag3!=-1) and (flag2!=flag3):
                tempr=matching_dict2[n][0]
                matching_dict2[n]=[flag2]
                matching_dict2[flag2]=[n]
                matching_dict2[tempr]=[flag3]
                matching_dict2[flag3]=[tempr]
        else:
            for k in adj_dict3[n]:
                if not matching_dict2[k]:
                    matching_dict2[n]=[k]
                    matching_dict2[k]=[n]
                    break


    yo1=0
    for i in range(0,num_nodes):
        if matching_dict2[i] and [i]==matching_dict2[matching_dict2[i][0]]:
            yo1=yo1+1
    greedy_size=yo1/2
    return greedy_size





def General():

################################INPUTS###############################################
    wb = xlwt.Workbook()
    ws = wb.add_sheet('Results_Random')
    book=xlrd.open_workbook('filenames_random.xls')
    style0 = xlwt.easyxf('font:bold on')
    style1 = xlwt.easyxf(num_format_str='#,###0.000')
    ###############################INPUTS############################################
    ws.write(0, 0,"File" ,style0)
    ws.write(0, 1, "Nodes",style0)
    ws.write(0, 2, "Initial_matching",style0)
    ws.write(0, 3, "NodesG",style0)
    ws.write(0, 4, "EdgesG",style0)
    ws.write(0, 5, "MaxDegreeG",style0)
    ws.write(0, 6, "NodesG_hat",style0)
    ws.write(0, 7, "EdgesG_Hat",style0)
    ws.write(0, 8, "Useless_edges",style0)
    ws.write(0, 9, "Forced_edges",style0)
    ws.write(0, 10, "MaxDegreeG_Hat",style0)
    ws.write(0, 11, "Classical_matching",style0)
    ws.write(0, 12, "Final_matching",style0)
    ws.write(0, 13, "Greedy_matching",style0)
    ws.write(0, 14, "Time_classical",style0)
    ws.write(0, 15, "Time_algorithm",style0)
    ws.write(0, 16, "Time_greedy",style0)
    ws.write(0, 17, "Perfect_matching_ratio",style0)
    ws.write(0, 18, "Nodes_ratio",style0)
    ws.write(0, 19, "Edges_ratio",style0)
    ws.write(0, 20, "Matching_ratio",style0)

    print "\nWelcome!!!This is an implementation for general random graphs. It uses Ed

    for index in range (1,481):#was 481

        if (index==482):
            f=open("Algotel_general.txt",'r').readlines()
        else:
            filea=str(book.sheet_by_index(0).cell(index,0))
            s=filea.replace("text:u","").replace("'","")
            f=open(s,'r').readlines()#True
            ws.write(index, 0, s)
        print index


#####################################################################################
        num_nodes=int([x.strip().split() for x in f][0][0])

        num_matching=int([x.strip().split() for x in f][2][0])


        adj_temp = [x.strip().split() for x in f][4:4+num_nodes]
        adj=[map(int,R) for R in adj_temp]

        matching_temp=[x.strip().split() for x in f][4+num_nodes+1:4+2*num_nodes+1]
        matching1=[map(int,R) for R in matching_temp]


        adj_dict1={i:[ii for ii,x in enumerate(L) if x] for i,L in enumerate(adj)}
        adj_dict2={i:[ii for ii,x in enumerate(L) if x] for i,L in enumerate(adj)}
        adj_dict3={i:[ii for ii,x in enumerate(L) if x] for i,L in enumerate(adj)}


        matching_dict1={i:[ii for ii,x in enumerate(L) if x] for i,L in enumerate(matc
        matching_dict2={i:[ii for ii,x in enumerate(L) if x] for i,L in enumerate(matc
        #print adj_dict2
        #print matching_dict2
        ws.write(index, 1, num_nodes)
        ws.write(index, 2, num_matching)

        G = nx.Graph()

        useless= nx.Graph()
        forced = nx.Graph()


    ####Fill Graph
        for n in range(0,num_nodes):
            for k in adj_dict1[n]:
                G.add_edge(n,k)


        Gt=G.number_of_edges()
        ws.write(index, 3, G.number_of_nodes())
        ws.write(index, 4, G.number_of_edges())
        ws.write(index, 5, max(G.degree().values()))

####Compute Edmonds classical matching on G for comparison


        startB = timeit.default_timer()
        Classical_Matching=nx.max_weight_matching(G, maxcardinality=False)
        stopB = timeit.default_timer()

####Compute G^ nad M^
        startC = timeit.default_timer()
        useless_edges(num_nodes,adj_dict1,matching_dict1,useless,adj_dict2,G)
        forced_edges(G,adj_dict2,matching_dict1,num_nodes,forced)
        Final_Matching=nx.max_weight_matching(G, maxcardinality=False)
        stopC = timeit.default_timer()

##Greedy
        startG = timeit.default_timer()
        greedy_size=Greedy(num_nodes,adj_dict3,matching_dict2)
        stopG = timeit.default_timer()


        ws.write(index, 6, G.number_of_nodes())
        ws.write(index, 7, G.number_of_edges())
        ws.write(index, 8, useless.number_of_edges())
        ws.write(index, 9, forced.number_of_edges())
        if G.degree().values():
            ws.write(index, 10, max(G.degree().values()))
        else:
            ws.write(index, 10, 0)
        ws.write(index, 11, len(Classical_Matching)/2)
        ws.write(index, 12, len(Final_Matching)/2+forced.number_of_edges())
        ws.write(index, 13, greedy_size)
        ws.write(index, 14, stopB - startB,style1)
        ws.write(index, 15, stopC - startC,style1)
        ws.write(index, 16, stopG - startG,style1)
        ws.write(index, 17, (len(Final_Matching)/2+forced.number_of_edges())/(num_nodes
        ws.write(index, 18, G.number_of_nodes()/Gt,style1)
        ws.write(index, 19, G.number_of_edges()/Gt,style1)
        ws.write(index, 20, (len(Final_Matching)/2+forced.number_of_edges())/(len(Class

    wb.save('AA_Results_Random_Reyni_Large_Bipartite.xls')

####Main function. Here we must choose the input form of the data
if __name__ == '__main__':


    startA = timeit.default_timer()
    General()
    stopA = timeit.default_timer()

    print "\nThis program has terminated.\n\nRunning time for 480 files:", stopA-start
    print "\n\nPlease open AA_Results_Random_Reyni_Large.xls file to see the Results"
```

Plot.mat

```
clear all;
close all;
load('RN1L.mat');
load('RN1S.mat');
load('RN2L.mat');
load('RN2S.mat');
load('RN3L.mat');
load('RN3S.mat');

%% Plots
%%
h(1)=figure('Name','Histogram of Matching size difference between Constraint Algorith

RN1L_Greedydiff=RN1L_Final_matching-RN1L_Greedy_matching;
RN2L_Greedydiff=RN2L_Final_matching-RN2L_Greedy_matching;
RN3L_Greedydiff=RN3L_Final_matching-RN3L_Greedy_matching;
RN1S_Greedydiff=RN1S_Final_matching-RN1S_Greedy_matching;
RN2S_Greedydiff=RN2S_Final_matching-RN2S_Greedy_matching;
RN3S_Greedydiff=RN3S_Final_matching-RN3S_Greedy_matching;

RN1L_Finaldiff=RN1L_Classical_matching-RN1L_Final_matching;
RN2L_Finaldiff=RN2L_Classical_matching-RN2L_Final_matching;
RN3L_Finaldiff=RN3L_Classical_matching-RN3L_Final_matching;
RN1S_Finaldiff=RN1S_Classical_matching-RN1S_Final_matching;
RN2S_Finaldiff=RN2S_Classical_matching-RN2S_Final_matching;
RN3S_Finaldiff=RN3S_Classical_matching-RN3S_Final_matching;

RN1L_Classicaldiff=(RN1L_Nodes./2)-RN1L_Classical_matching;
RN2L_Classicaldiff=(RN2L_Nodes./2)-RN2L_Classical_matching;
RN3L_Classicaldiff=(RN3L_Nodes./2)-RN3L_Classical_matching;
RN1S_Classicaldiff=(RN1S_Nodes./2)-RN1S_Classical_matching;
RN2S_Classicaldiff=(RN2S_Nodes./2)-RN2S_Classical_matching;
RN3S_Classicaldiff=(RN3S_Nodes./2)-RN3S_Classical_matching;

subplot(2,3,1)
hold on;
histf(RN1L_Greedydiff,0:1:10,'facecolor','b','facealpha',.9,'edgealpha',1)
histf(RN1L_Finaldiff,0:1:10,'facecolor','r','facealpha',.5,'edgealpha',1)
histf(RN1L_Classicaldiff,0:1:10,'facecolor','g','facealpha',.5,'edgealpha',1)
legalpha('Con-Gre','Clas-Con','Perf-Class','location','northeast')
legend boxoff
axis tight
xlabel('#edges') % x-axis label
ylabel('#instances') % y-axis label
title('RN1L')
grid on;

subplot(2,3,2)
hold on;
histf(RN2L_Greedydiff,0:1:10,'facecolor','b','facealpha',.9,'edgealpha',1)
histf(RN2L_Finaldiff,0:1:10,'facecolor','r','facealpha',.5,'edgealpha',1)
histf(RN2L_Classicaldiff,0:1:10,'facecolor','g','facealpha',.5,'edgealpha',1)
legalpha('Con-Gre','Clas-Con','Perf-Class','location','northeast')
legend boxoff
axis tight
xlabel('#edges') % x-axis label
ylabel('#instances') % y-axis label
title('RN2L')
grid on;

subplot(2,3,3)
hold on;
histf(RN3L_Greedydiff,0:1:10,'facecolor','b','facealpha',.5,'edgealpha',1)
histf(RN3L_Finaldiff,0:1:10,'facecolor','r','facealpha',.5,'edgealpha',1)
histf(RN3L_Classicaldiff,0:1:10,'facecolor','g','facealpha',.5,'edgealpha',1)
legalpha('Con-Gre','Clas-Con','Perf-Class','location','northeast')
legend boxoff
axis tight
xlabel('#edges') % x-axis label
ylabel('#instances') % y-axis label
title('RN3L')
grid on;

subplot(2,3,4)
hold on;
histf(RN1S_Greedydiff,0:1:10,'facecolor','b','facealpha',.5,'edgealpha',1)
histf(RN1S_Finaldiff,0:1:10,'facecolor','r','facealpha',.5,'edgealpha',1)
histf(RN1S_Classicaldiff,0:1:10,'facecolor','g','facealpha',.5,'edgealpha',1)
legalpha('Con-Gre','Clas-Con','Perf-Class','location','northeast')
legend boxoff
axis tight
xlabel('#edges') % x-axis label
ylabel('#instances') % y-axis label
title('R1NS')
grid on;

subplot(2,3,5)
hold on;
histf(RN2S_Greedydiff,0:1:10,'facecolor','b','facealpha',.5,'edgealpha',1)
histf(RN2S_Finaldiff,0:1:10,'facecolor','r','facealpha',.5,'edgealpha',1)
histf(RN2S_Classicaldiff,0:1:10,'facecolor','g','facealpha',.5,'edgealpha',1)
legalpha('Con-Gre','Clas-Con','Perf-Class','location','northeast')
legend boxoff
axis tight
xlabel('#edges') % x-axis label
ylabel('#instances') % y-axis label
title('RN2S')
grid on;

subplot(2,3,6)
hold on;
histf(RN3S_Greedydiff,0:1:10,'facecolor','b','facealpha',.5,'edgealpha',1)
histf(RN3S_Finaldiff,0:1:10,'facecolor','r','facealpha',.5,'edgealpha',1)
histf(RN3S_Classicaldiff,0:1:10,'facecolor','g','facealpha',.5,'edgealpha',1)
```

```
legalpha('Con-Gre','Clas-Con','Perf-Class','location','northeast')
legend boxoff
axis tight
grid on;
title('RN3S')
xlabel('#edges') % x-axis label
ylabel('#instances') % y-axis label
set(gcf,'PaperUnits','centimeters','PaperPosition',[0 0 50 30])
print('-dpng', 'RN1.png', '-r300');
%%
h(2)=figure('Name','Histogram of Max Degree of G AND G_Hat','NumberTitle','off')
RN1L_x=RN1L_MaxDegreeG-RN1L_MaxDegreeG_Hat;
RN2L_x=RN2L_MaxDegreeG-RN2L_MaxDegreeG_Hat;
RN3L_x=RN3L_MaxDegreeG-RN3L_MaxDegreeG_Hat;
RN1S_x=RN1S_MaxDegreeG-RN1S_MaxDegreeG_Hat;
RN2S_x=RN2S_MaxDegreeG-RN2S_MaxDegreeG_Hat;
RN3S_x=RN3S_MaxDegreeG-RN3S_MaxDegreeG_Hat;
subplot(2,3,1)
hold on;
histf(RN1L_MaxDegreeG,0:1:50,'facecolor','b','facealpha',.5,'edgealpha',1)
histf(RN1L_MaxDegreeG_Hat,0:1:50,'facecolor','r','facealpha',.5,'edgealpha',1)
histf(RN1L_x,0:1:50,'facecolor','c','facealpha',.5,'edgealpha',1)
legalpha('G','GHat','diff','location','northeast')
legend boxoff
axis tight
xlabel('MAX #edges') % x-axis label
ylabel('#instances') % y-axis label
title('RN1L')
grid on;

subplot(2,3,2)
hold on;
histf(RN2L_MaxDegreeG,0:10:500,'facecolor','b','facealpha',.5,'edgealpha',1)
histf(RN2L_MaxDegreeG_Hat,0:10:500,'facecolor','r','facealpha',.5,'edgealpha',1)
histf(RN2L_x,0:10:500,'facecolor','c','facealpha',.5,'edgealpha',1)
legalpha('G','GHat','diff','location','northeast')
legend boxoff
axis tight
xlabel('MAX #edges') % x-axis label
ylabel('#instances') % y-axis label
title('RN2L')
grid on;

subplot(2,3,3)
hold on;
histf(RN3L_MaxDegreeG,0:10:300,'facecolor','b','facealpha',.5,'edgealpha',1)
histf(RN3L_MaxDegreeG_Hat,0:10:300,'facecolor','r','facealpha',.5,'edgealpha',1)
histf(RN3L_x,0:10:300,'facecolor','c','facealpha',.5,'edgealpha',1)
legalpha('G','GHat','diff','location','northeast')
legend boxoff
axis tight
xlabel('MAX #edges') % x-axis label
ylabel('#instances') % y-axis label
title('RN3L')
```

```
grid on;

subplot(2,3,4)
hold on;
histf(RN1S_MaxDegreeG,0:1:50,'facecolor','b','facealpha',.5,'edgealpha',1)
histf(RN1S_MaxDegreeG_Hat,0:1:50,'facecolor','r','facealpha',.5,'edgealpha',1)
histf(RN1S_x,0:1:50,'facecolor','c','facealpha',.5,'edgealpha',1)
legalpha('G','GHat','diff','location','northeast')
legend boxoff
axis tight
xlabel('MAX #edges') % x-axis label
ylabel('#instances') % y-axis label
title('R1NS')
grid on;

subplot(2,3,5)
hold on;
histf(RN2S_MaxDegreeG,0:10:500,'facecolor','b','facealpha',.5,'edgealpha',1)
histf(RN2S_MaxDegreeG_Hat,0:10:500,'facecolor','r','facealpha',.5,'edgealpha',1)
histf(RN2S_x,0:10:500,'facecolor','c','facealpha',.5,'edgealpha',1)
legalpha('G','GHat','diff','location','northeast')
legend boxoff
axis tight
xlabel('MAX #edges') % x-axis label
ylabel('#instances') % y-axis label
title('RN2S')
grid on;

subplot(2,3,6)
hold on;
histf(RN3S_MaxDegreeG,0:5:300,'facecolor','b','facealpha',.5,'edgealpha',1)
histf(RN3S_MaxDegreeG_Hat,0:5:300,'facecolor','r','facealpha',.5,'edgealpha',1)
histf(RN3S_x,0:5:300,'facecolor','c','facealpha',.5,'edgealpha',1)
legalpha('G','GHat','diff','location','northeast')
legend boxoff
axis tight
grid on;
title('RN3S')
xlabel('MAX #edges') % x-axis label
ylabel('#instances') % y-axis label
set(gcf,'PaperUnits','centimeters','PaperPosition',[0 0 50 30])
print('-dpng', 'RN2.png', '-r300');



%%
h(3)=figure('Name','Computation time over # edges','NumberTitle','off')

subplot(2,3,1)
hold on;
stem(RN1L_EdgesG,RN1L_Time_classical,'g*')
stem(RN1L_EdgesG,RN1L_Time_algorithm,'ro')
stem(RN1L_EdgesG,RN1L_Time_greedy,'bx')
grid on;
```

```
title('RN1L')
xlabel('#Edges of G') % x-axis label
ylabel('Computation time (s)')% y-axis label
legend('Classical','Constraint','Greedy','Location','northoutside','Orientation','hor

subplot(2,3,2)
hold on;
stem(RN2L_EdgesG,RN2L_Time_classical,'g*')
stem(RN2L_EdgesG,RN2L_Time_algorithm,'ro')
stem(RN2L_EdgesG,RN2L_Time_greedy,'bx')
grid on;
title('RN2L')
xlabel('#Edges of G') % x-axis label
ylabel('Computation time (s)')% y-axis label
legend('Classical','Constraint','Greedy','Location','northoutside','Orientation','hor

subplot(2,3,3)
hold on;
stem(RN3L_EdgesG,RN3L_Time_classical,'g*')
stem(RN3L_EdgesG,RN3L_Time_algorithm,'ro')
stem(RN3L_EdgesG,RN3L_Time_greedy,'bx')
grid on;
title('RN3L')
xlabel('#Edges of G') % x-axis label
ylabel('Computation time (s)')% y-axis label
legend('Classical','Constraint','Greedy','Location','northoutside','Orientation','hor

subplot(2,3,4)
hold on;
stem(RN1S_EdgesG,RN1S_Time_classical,'g*')
stem(RN1S_EdgesG,RN1S_Time_algorithm,'ro')
stem(RN1S_EdgesG,RN1S_Time_greedy,'bx')
grid on;
title('RN1S')
xlabel('#Edges of G') % x-axis label
ylabel('Computation time (s)')% y-axis label
legend('Classical','Constraint','Greedy','Location','northoutside','Orientation','hor

subplot(2,3,5)
hold on;
stem(RN2S_EdgesG,RN2S_Time_classical,'g*')
stem(RN2S_EdgesG,RN2S_Time_algorithm,'ro')
stem(RN2S_EdgesG,RN2S_Time_greedy,'bx')
grid on;
title('RN2S')
xlabel('#Edges of G') % x-axis label
ylabel('Computation time (s)')% y-axis label
legend('Classical','Constraint','Greedy','Location','northoutside','Orientation','hor

subplot(2,3,6)
hold on;
stem(RN3S_EdgesG,RN3S_Time_classical,'g*')
stem(RN3S_EdgesG,RN3S_Time_algorithm,'ro')
stem(RN3S_EdgesG,RN3S_Time_greedy,'bx')
```

```
grid on;
title('RN3S')
xlabel('#Edges of G') % x−axis label
ylabel('Computation time (s)')% y−axis label
legend('Classical','Constraint','Greedy','Location','northoutside','Orientation','hor
set(gcf,'PaperUnits','centimeters','PaperPosition',[0 0 50 30])
print('−dpng', 'RN3.png', '−r300');


%%
h(4)=figure('Name','Sorted matching size','NumberTitle','off')
RN1L_Sorted_matching_temp(:,1)=RN1L_Classical_matching;
RN1L_Sorted_matching_temp(:,2)=RN1L_Final_matching;
RN1L_Sorted_matching_temp(:,3)=RN1L_Greedy_matching;
RN1L_Sorted_matching_temp(:,4)=RN1L_Initial_matching;
RN1L_Sorted_matching=sortrows(RN1L_Sorted_matching_temp);

RN2L_Sorted_matching_temp(:,1)=RN2L_Classical_matching;
RN2L_Sorted_matching_temp(:,2)=RN2L_Final_matching;
RN2L_Sorted_matching_temp(:,3)=RN2L_Greedy_matching;
RN2L_Sorted_matching_temp(:,4)=RN2L_Initial_matching;
RN2L_Sorted_matching=sortrows(RN2L_Sorted_matching_temp);

RN3L_Sorted_matching_temp(:,1)=RN3L_Classical_matching;
RN3L_Sorted_matching_temp(:,2)=RN3L_Final_matching;
RN3L_Sorted_matching_temp(:,3)=RN3L_Greedy_matching;
RN3L_Sorted_matching_temp(:,4)=RN3L_Initial_matching;
RN3L_Sorted_matching=sortrows(RN3L_Sorted_matching_temp);

RN1S_Sorted_matching_temp(:,1)=RN1S_Classical_matching;
RN1S_Sorted_matching_temp(:,2)=RN1S_Final_matching;
RN1S_Sorted_matching_temp(:,3)=RN1S_Greedy_matching;
RN1S_Sorted_matching_temp(:,4)=RN1S_Initial_matching;
RN1S_Sorted_matching=sortrows(RN1S_Sorted_matching_temp);

RN2S_Sorted_matching_temp(:,1)=RN2S_Classical_matching;
RN2S_Sorted_matching_temp(:,2)=RN2S_Final_matching;
RN2S_Sorted_matching_temp(:,3)=RN2S_Greedy_matching;
RN2S_Sorted_matching_temp(:,4)=RN2S_Initial_matching;
RN2S_Sorted_matching=sortrows(RN2S_Sorted_matching_temp);

RN3S_Sorted_matching_temp(:,1)=RN3S_Classical_matching;
RN3S_Sorted_matching_temp(:,2)=RN3S_Final_matching;
RN3S_Sorted_matching_temp(:,3)=RN3S_Greedy_matching;
RN3S_Sorted_matching_temp(:,4)=RN3S_Initial_matching;
RN3S_Sorted_matching=sortrows(RN3S_Sorted_matching_temp);


subplot(2,3,1)
hold on;
plot(RN1L_Sorted_matching(:,1),'b')
plot(RN1L_Sorted_matching(:,2),'g')
plot(RN1L_Sorted_matching(:,3),'r')
plot(RN1L_Sorted_matching(:,4),'m')
```

```
grid on;
title('RN1L')
xlabel('instances') % x—axis label
ylabel('size of matching')% y—axis label
legend('Classical','Constraint','Greedy','Initial','Location','northoutside','Orientat

subplot(2,3,2)
hold on;
plot(RN2L_Sorted_matching(:,1),'b')
plot(RN2L_Sorted_matching(:,2),'g')
plot(RN2L_Sorted_matching(:,3),'r')
plot(RN2L_Sorted_matching(:,4),'m')
grid on;
title('RN2L')
xlabel('instances') % x—axis label
ylabel('size of matching')% y—axis label
legend('Classical','Constraint','Greedy','Initial','Location','northoutside','Orientat


subplot(2,3,3)
hold on;
plot(RN3L_Sorted_matching(:,1),'b')
plot(RN3L_Sorted_matching(:,2),'g')
plot(RN3L_Sorted_matching(:,3),'r')
plot(RN3L_Sorted_matching(:,4),'m')
grid on;
title('RN3L')
xlabel('instances') % x—axis label
ylabel('size of matching')% y—axis label
legend('Classical','Constraint','Greedy','Initial','Location','northoutside','Orientat

subplot(2,3,4)
hold on;
plot(RN1S_Sorted_matching(:,1),'b')
plot(RN1S_Sorted_matching(:,2),'g')
plot(RN1S_Sorted_matching(:,3),'r')
plot(RN1S_Sorted_matching(:,4),'m')
grid on;
title('RN1S')
xlabel('instances') % x—axis label
ylabel('size of matching')% y—axis label
legend('Classical','Constraint','Greedy','Initial','Location','northoutside','Orientat

subplot(2,3,5)
hold on;
plot(RN2S_Sorted_matching(:,1),'b')
plot(RN2S_Sorted_matching(:,2),'g')
plot(RN2S_Sorted_matching(:,3),'r')
plot(RN2S_Sorted_matching(:,4),'m')
grid on;
title('RN2S')
xlabel('instances') % x—axis label
ylabel('size of matching')% y—axis label
legend('Classical','Constraint','Greedy','Initial','Location','northoutside','Orientat
subplot(2,3,6)
hold on;
plot(RN3S_Sorted_matching(:,1),'b')
plot(RN3S_Sorted_matching(:,2),'g')
plot(RN3S_Sorted_matching(:,3),'r')
plot(RN3S_Sorted_matching(:,4),'m')
grid on;
title('RN3S')
xlabel('instances') % x—axis label
ylabel('size of matching')% y—axis label
legend('Classical','Constraint','Greedy','Initial','Location','northoutside','Orientat
set(gcf,'PaperUnits','centimeters','PaperPosition',[0 0 50 30])
print('—dpng', 'RN4.png', '—r300');
%%
h(5)=figure('Name','Sorted Ratios','NumberTitle','off')
RN1L_ratio1=RN1L_NodesG_hat./RN1L_NodesG;
RN2L_ratio1=RN2L_NodesG_hat./RN2L_NodesG;
RN3L_ratio1=RN3L_NodesG_hat./RN3L_NodesG;
RN1S_ratio1=RN1S_NodesG_hat./RN1S_NodesG;
RN2S_ratio1=RN2S_NodesG_hat./RN2S_NodesG;
RN3S_ratio1=RN3S_NodesG_hat./RN3S_NodesG;

RN1L_ratio2=RN1L_EdgesG_Hat./RN1L_EdgesG;
RN2L_ratio2=RN2L_EdgesG_Hat./RN2L_EdgesG;
RN3L_ratio2=RN3L_EdgesG_Hat./RN3L_EdgesG;
RN1S_ratio2=RN1S_EdgesG_Hat./RN1S_EdgesG;
RN2S_ratio2=RN2S_EdgesG_Hat./RN2S_EdgesG;
RN3S_ratio2=RN3S_EdgesG_Hat./RN3S_EdgesG;

RN1L_ratio3=RN1L_MaxDegreeG_Hat./RN1L_MaxDegreeG;
RN2L_ratio3=RN2L_MaxDegreeG_Hat./RN2L_MaxDegreeG;
RN3L_ratio3=RN3L_MaxDegreeG_Hat./RN3L_MaxDegreeG;
RN1S_ratio3=RN1S_MaxDegreeG_Hat./RN1S_MaxDegreeG;
RN2S_ratio3=RN2S_MaxDegreeG_Hat./RN2S_MaxDegreeG;
RN3S_ratio3=RN3S_MaxDegreeG_Hat./RN3S_MaxDegreeG;

RN1L_ratio(:,1)=RN1L_ratio1;
RN1L_ratio(:,2)=RN1L_ratio2;
RN1L_ratio(:,3)=RN1L_ratio3;

RN2L_ratio(:,1)=RN2L_ratio1;
RN2L_ratio(:,2)=RN2L_ratio2;
RN2L_ratio(:,3)=RN2L_ratio3;

RN3L_ratio(:,1)=RN3L_ratio1;
RN3L_ratio(:,2)=RN3L_ratio2;
RN3L_ratio(:,3)=RN3L_ratio3;

RN1S_ratio(:,1)=RN1S_ratio1;
RN1S_ratio(:,2)=RN1S_ratio2;
RN1S_ratio(:,3)=RN1S_ratio3;

RN2S_ratio(:,1)=RN2S_ratio1;
```

```
RN2S_ratio(:,2)=RN2S_ratio2;
RN2S_ratio(:,3)=RN2S_ratio3;

RN3S_ratio(:,1)=RN3S_ratio1;
RN3S_ratio(:,2)=RN3S_ratio2;
RN3S_ratio(:,3)=RN3S_ratio3;

RN1L_ratio_sorted=sortrows(RN1L_ratio);
RN2L_ratio_sorted=sortrows(RN2L_ratio);
RN3L_ratio_sorted=sortrows(RN3L_ratio);
RN1S_ratio_sorted=sortrows(RN1S_ratio);
RN2S_ratio_sorted=sortrows(RN2S_ratio);
RN3S_ratio_sorted=sortrows(RN3S_ratio);


subplot(2,3,1)
hold on;
plot(RN1L_ratio_sorted(:,1),'b')
plot(RN1L_ratio_sorted(:,2),'g')
plot(RN1L_ratio_sorted(:,3),'r')
grid on;
title('RN1L')
ylabel('Nodes Ratio') % x-axis label
xlabel('Instances')% y-axis label
legend('Nodes','Edges','Max Degree','Location','northoutside','Orientation','horizonta

subplot(2,3,2)
hold on;
plot(RN2L_ratio_sorted(:,1),'b')
plot(RN2L_ratio_sorted(:,2),'g')
plot(RN2L_ratio_sorted(:,3),'r')
grid on;
title('RN2L')
ylabel('Nodes Ratio') % x-axis label
xlabel('Instances')% y-axis label
legend('Nodes','Edges','Max Degree','Location','northoutside','Orientation','horizonta

subplot(2,3,3)
hold on;
plot(RN3L_ratio_sorted(:,1),'b')
plot(RN3L_ratio_sorted(:,2),'g')
plot(RN3L_ratio_sorted(:,3),'r')
grid on;
title('RN3L')
ylabel('Nodes Ratio') % x-axis label
xlabel('Instances')% y-axis label
legend('Nodes','Edges','Max Degree','Location','northoutside','Orientation','horizonta

subplot(2,3,4)
hold on;
plot(RN1S_ratio_sorted(:,1),'b')
plot(RN1S_ratio_sorted(:,2),'g')
plot(RN1S_ratio_sorted(:,3),'r')
grid on;
```

```
title('RN1S')
ylabel('Nodes Ratio') % x—axis label
xlabel('Instances')% y—axis label
legend('Nodes','Edges','Max Degree','Location','northoutside','Orientation','horizonta

subplot(2,3,5)
hold on;
plot(RN2S_ratio_sorted(:,1),'b')
plot(RN2S_ratio_sorted(:,2),'g')
plot(RN2S_ratio_sorted(:,3),'r')
grid on;
title('RN2S')
ylabel('Nodes Ratio') % x—axis label
xlabel('Instances')% y—axis label
legend('Nodes','Edges','Max Degree','Location','northoutside','Orientation','horizonta

subplot(2,3,6)
hold on;
plot(RN3S_ratio_sorted(:,1),'b')
plot(RN3S_ratio_sorted(:,2),'g')
plot(RN3S_ratio_sorted(:,3),'r')
grid on;
title('RN3S')
ylabel('Nodes Ratio') % x—axis label
xlabel('Instances')% y—axis label
legend('Nodes','Edges','Max Degree','Location','northoutside','Orientation','horizonta
set(gcf,'PaperUnits','centimeters','PaperPosition',[0 0 50 30])
print('—dpng', 'RN5.png', '—r300');

%%
h(6)=figure('Name','Sorted #edges over #nodes','NumberTitle','off')

RN1L_DegreeG1=(RN1L_EdgesG)./(RN1L_NodesG);
RN1L_DegreeG_Hat1=(RN1L_EdgesG_Hat)./(RN1L_NodesG_hat);

RN1L_Degree_temp1(:,1)=RN1L_DegreeG1;
RN1L_Degree_temp1(:,2)=RN1L_DegreeG_Hat1;
RN1L_Degree_sorted1=sortrows(RN1L_Degree_temp1);
subplot(2,3,1)
hold on;

plot(RN1L_Degree_sorted1(:,1),'b')
plot(RN1L_Degree_sorted1(:,2),'g')
grid on;
title('RN1L')
ylabel('Graph density') % x—axis label
xlabel('Instances')% y—axis label
legend('G','G hat','Location','northoutside','Orientation','horizontal')


RN2L_DegreeG1=(RN2L_EdgesG)./(RN2L_NodesG);
RN2L_DegreeG_Hat1=(RN2L_EdgesG_Hat)./(RN2L_NodesG_hat);

RN2L_Degree_temp1(:,1)=RN2L_DegreeG1;
```

```
RN2L_Degree_temp1(:,2)=RN2L_DegreeG_Hat1;
RN2L_Degree_sorted1=sortrows(RN2L_Degree_temp1);


subplot(2,3,2)
hold on;
plot(RN2L_Degree_sorted1(:,1),'b')
plot(RN2L_Degree_sorted1(:,2),'g')
grid on;
title('RN2L')
ylabel('Graph density') % x-axis label
xlabel('Instances')% y-axis label
legend('G','G hat','Location','northoutside','Orientation','horizontal')

RN3L_DegreeG1=(RN3L_EdgesG)./(RN3L_NodesG);
RN3L_DegreeG_Hat1=(RN3L_EdgesG_Hat)./(RN3L_NodesG_hat);

RN3L_Degree_temp1(:,1)=RN3L_DegreeG1;
RN3L_Degree_temp1(:,2)=RN3L_DegreeG_Hat1;
RN3L_Degree_sorted1=sortrows(RN3L_Degree_temp1);


subplot(2,3,3)
hold on;
plot(RN3L_Degree_sorted1(:,1),'b')
plot(RN3L_Degree_sorted1(:,2),'g')
grid on;
title('RN3L')
ylabel('Graph density') % x-axis label
xlabel('Instances')% y-axis label
legend('G','G hat','Location','northoutside','Orientation','horizontal')

RN1S_DegreeG1=(RN1S_EdgesG)./(RN1S_NodesG);
RN1S_DegreeG_Hat1=(RN1S_EdgesG_Hat)./(RN1S_NodesG_hat);

RN1S_Degree_temp1(:,1)=RN1S_DegreeG1;
RN1S_Degree_temp1(:,2)=RN1S_DegreeG_Hat1;
RN1S_Degree_sorted1=sortrows(RN1S_Degree_temp1);


subplot(2,3,4)
hold on;
plot(RN1S_Degree_sorted1(:,1),'b')
plot(RN1S_Degree_sorted1(:,2),'g')
grid on;
title('RN1S')
ylabel('Graph density') % x-axis label
xlabel('Instances')% y-axis label
legend('G','G hat','Location','northoutside','Orientation','horizontal')

RN2S_DegreeG1=(RN2S_EdgesG)./(RN2S_NodesG);
RN2S_DegreeG_Hat1=(RN2S_EdgesG_Hat)./(RN2S_NodesG_hat);

RN2S_Degree_temp1(:,1)=RN2S_DegreeG1;
```

```
RN2S_Degree_temp1(:,2)=RN2S_DegreeG_Hat1;
RN2S_Degree_sorted1=sortrows(RN2S_Degree_temp1);


subplot(2,3,5)
hold on;
plot(RN2S_Degree_sorted1(:,1),'b')
plot(RN2S_Degree_sorted1(:,2),'g')
grid on;
title('RN2S')
ylabel('Graph density') % x—axis label
xlabel('Instances')% y—axis label
legend('G','G hat','Location','northoutside','Orientation','horizontal')

RN3S_DegreeG1=(RN3S_EdgesG)./(RN3S_NodesG);
RN3S_DegreeG_Hat1=(RN3S_EdgesG_Hat)./(RN3S_NodesG_hat);

RN3S_Degree_temp1(:,1)=RN3S_DegreeG1;
RN3S_Degree_temp1(:,2)=RN3S_DegreeG_Hat1;
RN3S_Degree_sorted1=sortrows(RN3S_Degree_temp1);


subplot(2,3,6)
hold on;
plot(RN3S_Degree_sorted1(:,1),'b')
plot(RN3S_Degree_sorted1(:,2),'g')
grid on;
title('RN3S')
ylabel('Graph density') % x—axis label
xlabel('Instances')% y—axis label
legend('G','G hat','Location','northoutside','Orientation','horizontal')
set(gcf,'PaperUnits','centimeters','PaperPosition',[0 0 50 30])
print('—dpng', 'RN6.png', '—r300');
%%
h(7)=figure('Name','Sorted mATCHED DEGREE','NumberTitle','off')
RN1L_ratioMax(:,1)=RN1L_MaxDegreeG;
RN1L_ratioMax(:,2)=RN1L_MaxDegreeG_Hat;

RN1L_ratio_sortedMax=sortrows(RN1L_ratioMax);
subplot(2,3,1)
hold on;
plot(RN1L_ratio_sortedMax(:,1),'b')
plot(RN1L_ratio_sortedMax(:,2),'g')
grid on;
title('RN1L')
ylabel('Max Degree') % x—axis label
xlabel('Instances')% y—axis label
legend('Max degree G','Max degree G hat','Location','northoutside','Orientation','hor

RN2L_ratioMax(:,1)=RN2L_MaxDegreeG;
RN2L_ratioMax(:,2)=RN2L_MaxDegreeG_Hat;

RN2L_ratio_sortedMax=sortrows(RN2L_ratioMax);
subplot(2,3,2)
```

```
hold on;
plot(RN2L_ratio_sortedMax(:,1),'b')
plot(RN2L_ratio_sortedMax(:,2),'g')
grid on;
title('RN2L')
ylabel('Max Degree') % x-axis label
xlabel('Instances')% y-axis label
legend('Max degree G','Max degree G hat','Location','northoutside','Orientation','hor

RN3L_ratioMax(:,1)=RN3L_MaxDegreeG;
RN3L_ratioMax(:,2)=RN3L_MaxDegreeG_Hat;

RN3L_ratio_sortedMax=sortrows(RN3L_ratioMax);
subplot(2,3,3)
hold on;
plot(RN3L_ratio_sortedMax(:,1),'b')
plot(RN3L_ratio_sortedMax(:,2),'g')
grid on;
title('RN3L')
ylabel('Max Degree') % x-axis label
xlabel('Instances')% y-axis label
legend('Max degree G','Max degree G hat','Location','northoutside','Orientation','hor

RN1S_ratioMax(:,1)=RN1S_MaxDegreeG;
RN1S_ratioMax(:,2)=RN1S_MaxDegreeG_Hat;

RN1S_ratio_sortedMax=sortrows(RN1S_ratioMax);
subplot(2,3,4)
hold on;
plot(RN1S_ratio_sortedMax(:,1),'b')
plot(RN1S_ratio_sortedMax(:,2),'g')
grid on;
title('RN1S')
ylabel('Max Degree') % x-axis label
xlabel('Instances')% y-axis label
legend('Max degree G','Max degree G hat','Location','northoutside','Orientation','hor

RN2S_ratioMax(:,1)=RN2S_MaxDegreeG;
RN2S_ratioMax(:,2)=RN2S_MaxDegreeG_Hat;

RN2S_ratio_sortedMax=sortrows(RN2S_ratioMax);

subplot(2,3,5)
hold on;
plot(RN2S_ratio_sortedMax(:,1),'b')
plot(RN2S_ratio_sortedMax(:,2),'g')
grid on;
title('RN2S')
ylabel('Max Degree') % x-axis label
xlabel('Instances')% y-axis label
legend('Max degree G','Max degree G hat','Location','northoutside','Orientation','hor

RN3S_ratioMax(:,1)=RN3S_MaxDegreeG;
RN3S_ratioMax(:,2)=RN3S_MaxDegreeG_Hat;
```

```
RN3S_ratio_sortedMax=sortrows(RN3S_ratioMax);
subplot(2,3,6)
hold on;
plot(RN3S_ratio_sortedMax(:,1),'b')
plot(RN3S_ratio_sortedMax(:,2),'g')
grid on;
title('RN1L')
ylabel('Max Degree') % x—axis label
xlabel('Instances')% y—axis label
legend('Max degree G','Max degree G hat','Location','northoutside','Orientation','hor
set(gcf,'PaperUnits','centimeters','PaperPosition',[0 0 50 30])
print('—dpng', 'RN7.png', '—r300');


%tilefigs()%(h)
```

Bibliography

[1] C. Berge, “TWO THEOREMS IN GRAPH THEORY,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 43, no. 9, pp. 842–844, Sep. 1957.

[2] L. Bui, S. Sanghavi, and R. Srikant, “Distributed link scheduling with constant overhead,” *IEEE/ACM Trans. Netw.*, vol. 17, no. 5, pp. 1467–1480, 2009.

[3] R. Duan and S. Pettie, “Linear-time approximation for maximum weight matching,” *J. ACM*, vol. 61, no. 1, 2014.

[4] J. Edmonds, “Paths, trees, and flowers,” *Canadian Journal of Mathematics*, vol. 17, pp. 449–467, 1965.

[5] J. E. Hopcroft and R. M. Karp, “An n$^{5/2}$ algorithm for maximum matchings in bipartite graphs,” *SIAM J. Comput.*, vol. 2, no. 4, pp. 225–231, 1973.

[6] S. Micali and V. V. Vazirani, “An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs,” in *21st Symp. on Foundations of Comp. Sc. (FOCS)*. IEEE, 1980, pp. 17–27.

[7] N. Nisse, A. Salch, and V. Weber, “Recovery of disrupted airline operations,” 2015, http://hal.inria.fr/Something.

[8] X. Wu and R. Srikant, “Regulated maximal matching: A distributed scheduling algorithm for multi-hop wireless networks with nodeexclusive spectrum sharing,” in *IEEE Conf. on Decision and Control*, 2005.