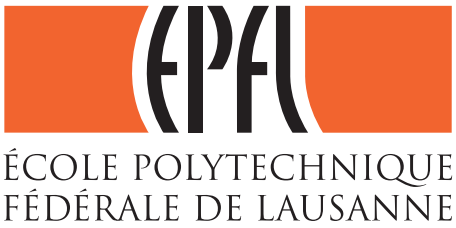


Semester Project

An Maxflow Algorithm for Directed Planar Graphs

Supervisor:
Dr. Andreas Karrenbauer
andreas.karrenbauer@epfl.ch

Author:
Luca Furrer
luca.furrer@epfl.ch

Chair of Discrete Optimization (Prof. F. Eisenbrand)

January 2010

Abstract

In this paper we present the maximum flow algorithm in $O(n \log n)$ for directed planar graphs of Glencore Borradail and Philip Klein published in 2009 in [1]. Further we are going to present and analyse an implementation of this algorithm.

To analyse the maximum flow algorithm we also use an algorithm to find the maximal running time of a an algorithm for planar graphs which is based on transforming maximal planar graphs to other maximal planar graphs by Lawson flips. This algorithm will be presented and analyzed as well.

Contents

1 Introduction

In this paper we are going to discuss an maximum flow algorithm for planar flow which has been introduced by Glencora Borradaile and Philip Klein in [1]. This algorithm is running in $O(n \log n)$. Further we are going to analyze an implementation of this algorithm.

This paper results out of an semester project at the Ecole Polytechnique fédérale de Lausanne (EPFL) under supervision of Dr. Andreas Karrenbauer of the Chair of Discrete Optimization of Prof. Friedrich Eisenbrand.

The problem of finding a maximum flow algorithm is by history connected to planar graphs as it is discussed by Schrijver in [8] and it appeared in around 1955 in researched lied to the cold war. A development of the running time for maximum flow algorithm is present in Schrijvers textbook [9, p.160-161]. In 1997 Karsten Weihe presented an $O(n \log n)$ algorithm to find the maximal flow in a directed planar graph. However it seems that there is an small error in his proof as stated by Borradaile and Klein in [1, p.9:4], which also have published an $O(n \log n)$ algorithm for directed planar graph in this paper.

So in this paper we present the algorithm of Borradaile and Klein and perform several tests on a implementation of this algorithm. So we will analyse the observed running time of the implementation using an algorithm to find the graph with the highest running time. But to do so we need first to introduce some theory about graphs and triangulations.

2 Graph Theory

In this section we introduce some graph theory which is needed to understand the following chapters of this report. We are mainly following Chapter 3 of Schrijver [9]. This book is recommended for further reading on graph theory since it gives a good overview over this topic.

Definition 1 *An* undirected graph *is a pair $G = (V, E)$, where V is a finite set and E is a set of unordered pairs of elements in V. The elements of V are called vertices, nodes or points. The elements of E are called edges.*

But somehow in this paper we are more interested in directed graphs.

Definition 2 *A* directed graph *or digraph is a pair $G = (V, A)$ where E is a finite set and A is a set of ordered pairs of elements in V. The elements of V are, as in a undirected graph. called vertices, nodes or points. The elements of A are called arcs or directed edges.*

Definition 3 *A graph $G' = (V', A')$ is a subgraph of $G = (V, A)$ if $V' \subseteq V$ and $A' \subseteq A$.*

Definition 4 *Let $a = (u, v) \in A$ be an arc from u to v of the graph G. So $-a$ denotes the arc from v to u, i.e. $-a = (v, u)$. $-a$ is called the reverse arc. With $-A$ we denote the set of the reversed arcs of A.*

One may see that $\overline{G} = (V, A \cup -A)$ has the same properties as an undirected graph. Therefore it is possible to represent a undirected graph as a directed graph.

Often one may be interested which is the couple (u, v) which defines an arc.

Definition 5 *Let $a = (u, v) \in A$ be an arc. Then u is called tail of a and v is called head of a. We note $u = tail(a)$ and $v = head(a)$*

Definition 6 *We denote with $\delta(u)$, $u \in V$ the set of incident arcs of u, i.e. $\{a = (v, w) \in A | a \cap u \neq \emptyset\}$. We denote with $\delta^{in}(u)$, $\delta^{out}(u)$ the set of incoming respectively outgoing arcs of u.*

There are several important subsets of graphs which are needed to understand the problem of a maximal flow.

Definition 7 *A subset $\{a_1, \ldots, a_n\}$ of A is called walk if for all $1 < i < n$ $tail(a_i) = head(a_{i-1})$ and $head(a_i) = tail(a_{i+1})$.*

Definition 8 *A walk is called path every arc contained in the walk appears only once. A path from s to t is called an s-t-path.*

Definition 9 *A path $\{a_1, \ldots, a_n\}$ is called cycle if $a_1 = a_n$. A cycle is called simple if every node contained in the cycle is only head to one arc in the cycle.*

Definition 10 *Walks, path and cycles are said to be directed if all arcs have the same directions. Otherwise we call them undirected.*

Definition 11 *Two nodes u and v are called connected if there exists an undirected path between u and v.*

Definition 12 *A tree is a connected subgraph T of G without cycles. A tree containing all nodes of G is called spanning tree.*

Definition 13 *A graph G is called planar, if there exists a continuous injection of G to $\mathbb{R}^2$.*

This means that if we represent a graph G in the plane, it is possible to draw him in such manner that there are no edges crossing each other. The maximum s-t-flow algorithm presented later works only with a directed planar graph.

Definition 14 *A subdivision of a graph G is a graph obtained from G by inserting new vertices into the edges of G.*

Theorem 1 (Kuratowski's theorem) *A graph G is planar if and only if no subgraph of G is a subdivision of K_5 or $K_{3,3}$ (Figure 1).*

Definition 15 *The topological components of $\mathbb{R}^2 \backslash G$ are called faces. We denote the set of faces with F.*

There is also a combinatorial way to define the faces: From a starting node and arc, we will follow always the next arc in counter-clockwise direction until we get back to the starting node. The face is the area to the right of the path. We will also get the same result by choosing the next clockwise arc and considering the area to the right.

One face, normally the outer face, of a connected graph is called infinity face and denoted by f_∞. One may see, that depending on the graphical representation chosen for the graph, the outer face changes, so by denoting the infinity face we choose also which face is the outer face, but at the end it is arbitrary.

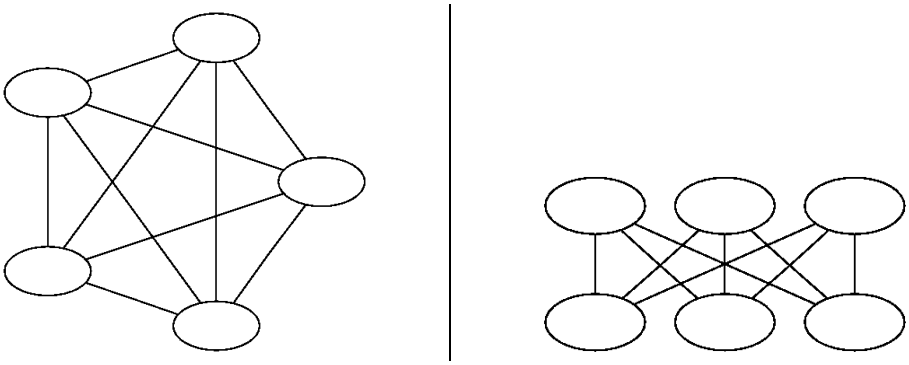


Figure 1: K_5 (left) and $K_{3,3}$ (right)

Theorem 2 (Euler's formula) *By Euler's formula, for every connected graph $G = (V, A)$ and its face collection F, it holds that*

$$|V| + |F| = |E| + 2.$$

Definition 16 *A graph is simple if it contains no arc $a = (u, v)$ $u, v \in V$ such that $u = v$ and there are no multiple arcs connecting the same pairs of nodes i.e $\nexists a = (u, v), a' = (u', v')$ $a, a' \in A$ such that $u = u'$ and $v = v'$.*

Definition 17 *A simple planar graph is called maximal if it is not possible to add an arc to A without violating the properties of a simple planar graph.*

It is important to see, that there is no unique maximal planar graph for a fixed number of nodes. The get an overview on the number of non-ismorphic undirected planar graphs we recommend to consult the article of G. Birkmann and B. D. McKay [2].

Lemma 1 *A maximal planar graph has only triangular faces.*

Since a maximal planar graphs have only triangular faces they is also often called *planar triangulations*.

Proof: Consider a simple planar graph $G = (V, A)$. Suppose that G is maximal. It is obvious that a face with only two nodes cannot appear in a simple graph. Suppose now that there exists a face with more than three nodes on the border. Hence there exists two nodes $u, v \in V$ such that there is no arc $(u, v) \in A$. Since u and v are at the same face it is possible to draw an arc a' between u and v without crossing an other arc. Hence $G' = (V, A \cup a')$ is a simple planar graph. This implies that G was not maximal. Therefore every face of G is triangular.

□

Theorem 3 *The maximal number of arcs for a planar graph with is* $|A| = 3|V| - 6$.

Proof: Consider the planar graph $G = (V, A)$. To simplify the notations we set $|A| = n$, $|V| = m$ and $|F| = f$.

To every arc there are two corresponding faces. Lemma 1 implies that $f = 3m$ for the maximal number of arcs, hence $2m = 3f$ By Euler's formula $m - n + f = 2$ we get

$$\begin{aligned} n - m + \frac{2}{3} &= 2 \\ 3n - 3e + 2e &= 6 \\ 3n - 6 &= e \end{aligned}$$

Therefore the maximal number of arcs in a planar graph is $3n - 6$.

□

Definition 18 *Given a directed planar graph G, the dual graph of G is the graph $D = (F, A)$ where F are the faces of G. There is an one to one connection between the arcs of the primal and the dual graph. To get the direction of the arcs in A one turn the arc in G around 90° clockwise.*

To detect the directions of the arcs in the dual graph it is possible to apply the left hand rule: Take your left hand with the palm downwards. If the index points in direction of the arc in the primal graph, the thumb points in direction of the arc in the dual graph.

Definition 19 *The function $c : A \to \mathbb{R} \times \mathbb{R}$, $c(a) = [l, u]$ is called capacity function. We define $c(-a) = -l(a)$ $\forall a \in A$*

In this paper we will often set $l = 0$ and denote $c(a) = u$. It is also possible to interpret this function as length function of the arc depending on the context.

Definition 20 *An s-t-flow f in a graph $G = (V, A)$ is a function $f : A \to \mathbb{R}$ such that*

1. $l(a) \leq f(a) \leq u(a) \forall a \in A$
2. $f(\delta^{in}(v)) = f(\delta^{out}(v)) \forall v \in V \backslash \{s, t\}$

The value of an s-t-flow f is defined by

$$value(f) = f(\delta^{out}(s)) - f(\delta^{in}(s)) = -f(\delta^{in}(t)) - f(\delta^{out}(s)).$$

We set $f(-a) = -f(a)$ for all $a \in A$.

We will sometime call an s-t-flow just flow for simplicity.

Definition 21 *The residual graph $G_f = (V, A_f)$ is defined by*

$$A_f = \{a \in A : f(a) < c(a)\} \cup \{-a \in -A : f(a) > l(a)\}.$$

The residual capacity function c_f is defined by

$$c_f(a) = c(a) - f(a) \quad \forall a \in A_f$$

Definition 22 *We define $excess_f(v) = f(\delta^{in}(v)) - f(\delta^{out}(v))$ for v in V.*

Definition 23 *A s-t-cut in a graph $G = (V, A)$ is a subset S of arcs such that there exists no path from s to t in the graph $(V, A \backslash S)$. The value of a cut is equal to $c(\delta^{out}(S))$.*

With this two last definitions the following important theorem makes sense.

Theorem 4 (Max-Flow-Min-Cut-Theorem) *The value of the minimal s-t-cut is equal to the value of the maximal s-t-flow.*

With this problem we know the dual problem to the maximal flow problem. It allows also to check if a given solution is optimal. The proof is omitted and can be found in every standard book about graph theory. Further connected to the definition of the s-t-flow it follows the definition of a circulation.

Definition 24 *A function $f : A \to \mathbb{R}$ is called circulation if $excess_f(v) = 0 \forall v \in V$.*

It can be seen as an s-t-flow with $s = t$.

Definition 25 *Let $G = (V, A)$ be a graph ant let $l : A \to \mathbb{R}$. A function $p : V \to \mathbb{R}$ is called a potential if for all $a = (u, v)$*

$$l(a) \geq p(v) - p(u)$$

We are going to use a potential function which was introduced by Miller and Naor [7, p.6]. In this case $l(a) = c(a)$ and the potential of a node is the length of the shortest path to this node from a fixed source node with, by convention, potential 0. We are using this potential mainly in the dual graph as for example in the following definition.

Definition 26 *A directed cycle is called clockwise if all included faces have non-negative potentials. In the same way we say that a cycle is called counterclockwise if the potential of every included face is non-positive.*

One may check that this definition corresponds to the movement of the hand of a clock. It is also possible that a cycle is neither clockwise nor counterclockwise. But on the other side a cycle is always clockwise or counterclockwise.

Definition 27 *A s-t-walk A is said left of an other s-t-walk B if the cycle $A - B$ is clockwise.*

In the same way we can also define that A right of B is the cycle is counterclockwise.

Definition 28 *A s-t-walk A is leftmost if for all other s-t-walk B, A is left of B*

In the later presented LeftMostFlow algorithm we use the definition of leftmost path quite extensively.

3 Triangulations

Since we will be interested later on to generate different maximal planar graphs we are going to consider triangulations and investigate a possibility to transform one transformation to an other.

We consider a finite set of points S in the euclidian two-dimensional space.

Definition 29 *The convex hull of a set of points is the smallest convex region containing the set.*

Definition 30 *An point in S is an extreme point if he is on the boundary of the convex hull and the interior angle two the next two points on the boundary is smaller than π.*

Definition 31 *A basic triangle is a non-degenerated triangle whose sides line segments not intersected by other segments and there is no element of except the three defining the triangle contained in the triangle.*

Definition 32 *If the convex hull of S is covered by non-overlapping basic triangles, this is called a triangulation of S.*

Definition 33 *We define a flip as the transformation of two triangles abd and bcd with common side bd to the triangles abc and acd with common side ac. Those flips are often called* Lawson Flips

Theorem 5 *Given any two triangulations of S, say T' and T'', there exists a (finite) sequence of flips by which T' can be transformed to T''*

This theorem is due to Lawson [6] in which he also introduces the flips, called exchanges, who appears later under the name of Lawson Flips.

The proof, which is omitted in this report, can be found either in the paper of Lawson mentioned before, or a shorter version in Felsner [4, p.119] which uses Delaunay triangulations. The main idea of both proofs is to fix a reference triangulation and to show that it is possible to transform every triangulation into this reference triangulation. Lawson uses constructs his reference triangulations from the outside of the convex set to the inside by removing extreme points. Felsner uses the Delaunay triangulation as reference triangulation. He shows as well that this triangulation can be reached in at most $\binom{n}{2}$ flips.

Corollary 1 *It is possible to transform every maximal planar graph by flipping edges to any other maximal planar graph with the same number of nodes.*

Proof: By Lemma 1 we know that every maximal planar graph has only triangular faces and hence it is also a triangulation. Hence by Theorem 5 we have that it is possible to reach every triangulation of a node set by flipping edges starting at one triangulation.

□

This corollary is going to be very useful in the section 5 since it allows us to transform one maximal planar graph to an other by flipping arcs. Therefore for every maximal planar graph there exists a set of flips to transform it to another maximal planar graph.

4 Maximum s-t-flow algorithm for planar graphs

In this section we preset the $O(n \log n)$ maximum s-t flow algorithm for planar graphs of Glencora Borradaile and Philip Klein [1].

For this algorithm we consider a directed planar graph $G = (V, A)$ with positive capacities $c : A \to \mathbb{R}^+$and given source s and sink t.

The algorithm called MaxFlow consists mainly out of two steps, one is LeftMostCirculation and the other is LeftMostPath.

Algorithm 1 (MaxFlow)

1. *Designate a face incident to s as f_∞*
2. *Saturate the clockwise cycles (LeftMostCirculation)*
3. *While there is a residual s-t-path, saturate the leftmost such path (LeftMostFlow)*

In the following sections the two sub-algorithm for Algorithm 1 will be presented. It follows a short discussion about the running time.

4.1 LeftMostCirculation

The second step of the MaxFlow algorithm is the LeftMostCirculation. It is due to S. Khuller, J. Naor and P. Klein in [5].

In this step we saturate all the clockwise cycles of the graph G. This is done by searching for the shortest path from f_∞ to the other faces in the dual graph and interpret this distance as node potentials. The flow over an arc is interpreted as the difference between the two nodes in the dual graph.

Let $D = (F, A)$ denote the dual graph of G. In order to avoid confusion between the notations for faces and the one for the flow. We use here $flow$ instead of f to denote the function $f : A \to \mathbb{R}$ for the circulation.

Algorithm 2 (LeftMostCirculation)

1. *Interpret capacities $c(a)$, $a \in A$ as length of the arc in the dual graph D.*
2. *Let $\Phi(f) = dist(f_\infty, f)$ for all $f \in F$.*
3. *Let $flow(a) = \Phi(head_D(a)) - \Phi(tail_D(a))$ for all $a \in A$*
4. *Return $flow$.*

The LeftMostCirculation algorithm presented in [5] and the one in [1] differ slightly. We presented here a version which follows [5].

At step 2 of Algorithm 2 one may for example use the Dijkstra algorithm to obtain the shortest path tree. This is possible since we have positive length. The Dijkstra algorithm can be found for example in Schrijver [9, p.97] or in de Werra, Liebling and Hêche [3, p.179]. The Dijkstra algorithm can have a running time of $O(n \log n + m)$.

Further a lemma due to Khuller Naor and Klein gives us that this algorithm really saturates clockwise cycles as claimed in Algorithm 1.

Lemma 2 *The graph has no clockwise cycle that is residual with respect to the circulation returned by LeftMostCirculation.*

Proof: Let C be a clockwise cycle. Let a be an arc in C which is in the shortest path tree computed in Step 2. Since every face enclosed to C is connected to f_∞ this arc exists. By definition of the residual capacity, we have

$$c_{flow}(a) = c(a) - flow(a).$$

By the definition of $flow$ in step 3 we get

$$\begin{aligned} c_{flow}(a) &= c(a) - \Phi(head_D(a)) - \Phi(tail_D(a)) \\ &= c(a) - (dist(f_\infty, head_D(a)) - dist(f_\infty, tail_D(a))) \\ &= c(a) - c(a) = 0. \end{aligned}$$

Therefore there is no clockwise cycle in the residual graph with respect to $flow$.

□

4.2 LeftMostFlow

The LeftMostFlow algorithm takes the circulation returned by LeftMostCirculation and is searching for the leftmost s-t-path in order to saturate it until there is no s-t-path left.

Algorithm 3 (LeftMostFlow)

1. *Initialize $f = flow$*
2. *While there is an s-t-path in G_f, saturate the leftmost residual s-t-path and modify f.*
3. *Return f*

To discuss the running time of Algorithm 3, we present first the following theorem.

Theorem 6 (Usability Theorem) *Consider the Algorithm 3. Suppose that an arc a is augmented and some time later his reverse arc is augmented. Then arc a cannot be augmented again.*

The proof of theorem can be found in the paper of Borradaile and Klein [1].

But even with this theorem it is not assured that the LeftMostFlow algorithm runs in the wanted $O(n \log n)$ time. For this reasen Borradaile and Klein present a more specific algorithm which runs in the wanted time.

Algorithm 4 (LeftMostFlow, specific)

1. *Initialize* $f = flow$
2. *Initialize* T *to be the right-first search tree searching backwards from* t *in* G_f
3. *Let* $\bar{G}$ *be the graph obtained by deleting all vertices not in* T
4. *Initialize* T^* *to consist of the egdes not in the dual* D *of* G_f *that are not in* T.
5. *Repeat:*
6. *If the s-t path in* T *is residual, saturate it, modifying* f.
7. *Let* d *be the last non-residual arc in the path from* s *to* t.
8. *If the tail of* d *in* D *is a descendent in* T^* *of the* $head(d)$ *in* D*; return* f
9. *Let* e *be the parent arc in* T^* *of* $head_D(d)$
10. *Eject* e *from* T^* *and insert* d *into* T^*.
11. *Eject* d *from* T *and insert* e *into* T.

Theorem 7 *Algorithm 4 takes* $O(n \log n)$ *time.*

The proof of Theorem 7 can be found in the article of Borradaile and Klein [1]. With this algorithm number of pushes over an arc is minimized

4.3 Running time

The running time of the MaxFlow algorithm is $O(n \log n)$. For that is true we must analyse the LeftMostCirculation as well as the LeftMostFlow algorithm.

What concerns the LeftMostCirculation algorithm, one may observe that the construction of the dual graph is not a problem of this algorithm. It is obvious that the limiting part in terms of running time is the computation of the shortest path tree which can, by choosing a appropriate algorithm be handeld in the $O(n \log n)$. The computation of $flow$ can be done in $O(m)$ which is, since we are in a planar graph bounden by $3n-6$ (Theorem 3). Hence the first part of the MaxFlow algorithm holds the wanted running time.

For the LeftMostFlow, by Theorem 7 Algorithm 4 has running time $O(n \log n)$. Hence the MaxFlow algorithm has running time $O(n \log n)$.

5 TestGraph algorithm

We are interested to find the graph and capacities for a graph such that it takes the longest time possible to run an algorithm in order to find the worst case graph and as well to check if the proposed upper bound for the running time is valid.

Since the number of different planar graph with a fixed number of vertices is increasing quite fast. It is impossible to test all different graphs, even without considering the problem of generating all of them. When even different capacities for the arcs are involved this gets impossible.

Hence one must find an other way to find this worst case graph. In this section we present one approach which we call the TestGraph algorithm.

The goal of this algorithm is to allow to test an directed planar graph algorithm and to find a worst case graph as well as its running time. In this case the tested algorithm will be the MaxFlow algorithm of section 4.

To test the algorithm we create a maximal planar graph G with n nodes since every planar graph with at most n nodes is a subgraph of G. Hence the graph with maximum running time is a subgraph of a maximal planar graph and can hence be obtained by setting the capacities of the arcs not included in the subgraph to zero. Further we use Corollary 1 to transform one maximal planar graph to an other.

Algorithm 5 (TestGraph)

1. *Create a maximal planar graph G.*
2. *For large $n \in \mathbb{N}$ do*
   (a) $G' \leftarrow Transform(G)$
   (b) *If running time of G' is higher than the one of G*
      - $G \leftarrow G'$
      - *If running time of G is bigger than the previous maximum. Stock G as graph with maximal running time.*
   (c) *Else:* $G \leftarrow Markov(G, G')$
3. *Return the graph with maximal running time.*

In Algorithm 5 in Step 2a we call Transform(G). This algorithm chooses one arc in the graph G and flips it as well as it changes its capacity. The flip corresponds to the flip of Definition 33. By Corollary 1 we know that the new graph is also maximal. Like this we transform G to an other maximal planar graph which differs from G only in one arc.

Algorithm 6 (Transform(G))

1. *Choose randomly an arc a of G*
2. *Flip or inverse arc a*
3. *Change capacity of arc a.*
4. *Return G*

As we change the capacity in Algorthim 6 in step 3, it is suggested to do this in a using a Random Walk with several steps and in each step probability 1/2 to go up and probability 1/2 to go down. Since we choose the arc over an uniform distribution we can apply some results of the coupon collector problem to estimate the number of iteration are needed, such that every arc is changed at least once. This number is given approximately by $m \log m$. Note that this number does not give an approximation of needed iterations since flipping every arc at least once does not automatically lead to the graph with maximal running time. But it shows already that the number it will not be enough to augment the number of iteration linearly to get a similar good resulat with a bigger number of nodes.

In TestGraph at step 2c, there is an Algorithm Markov(G,G') called. This algorithm is only called if the running time for G is higher than the one of G'. In this case we want sometimes to keep the previous graph since he has higher running time, but it might also by an advantage to change the graph to get an higher run time. The choice whether to change to an other graph G' or not is done by this algorithm.

Algorithm 7 (Markov(G,G'))

1. *Let E, E' be the running time of G resp G'*
2. *With probability $e^{\beta(E'-E)}$ return G', otherwise return G*

In step 2 we use a β it can be chosen such that the acception rate is bigger or smaller, as it is prefered.

One may observe that we get the following probability to change from G to G' in TestGraph:

$$P(G \to G') = \begin{cases} 1 & if E' > E \\ e^{\beta(E'-E)} & if E' \leq E \end{cases}$$

Further in one iteration of TestGraph the Markov property is fulfilled, that is the change made in iteration i is independent of iteration $i-1$. So we get finally a Markov-Chain-Monte-Carlo algorithm.

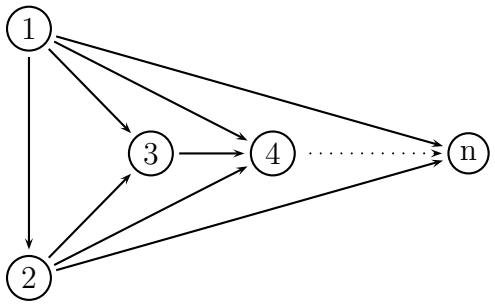
6 Implementation

In this section we are presenting some specifications for the implementation of MaxFlow and TestGraph algorithm (Algorithms 1 and 5). We will not discuss the the exact form of the implementation which can be found in the code documentation.

6.1 Graph classes

The class Directed_Planar_Graph and the class Annotated_Graph with the type definition for Annotated_Planar_Graph to add annotations to a graph are based on a code written by Martin Niemeier for directed graphs. To store the planarity of the graph there is embedding imposed which specifies the order of out and incoming edges of a node. This order is counterclockwise. To read in a graph a special data format is imposed. The format demands first the number of nodes followed the number of arcs of the graph followed by the arcs (tail head index). It is necessary to keep track also of reversed arcs for the embedding. For annotated graphs the annotations for the graphs follows after an * of the form (arc capacity flow). For an example see the files in Appendix A.1 and A.2 where a graph with 5 respectively 10 nodes is represented. It is possible to read out just the graph of a data structure for an annotated graph. Further it is not possible to modify the graph of an annotated graph and for this reason it is also not possible to copy them.

If a maximal planar graph is created he has initially the following form:



Hence if one wants an random maximal planar graph it is suggested to use the rndFlip function of maxgraph.h several time to obtain a random maximal planar graph.

6.2 MaxFlow algorithm

The implemented algorithm follows the algorithm described in section 4. In order to compute the shortest path tree in Algorithm 2, the dijkstra algorithm is used since the the capacities and therfore the length of the arcs arcs are positive.

For the LeftMostFlow algorithm, the implemented version is not the specific one (Algorithm 4) but Algorithm 3 with a simple depth first search to find the left most path. This is less efficient than the specific version but lets more space open for further improvements.

6.3 TestGraph algorithm

The implemented algorithm follows the algorithm TestGraph (Algorithm 5 of Section 5. We use $\beta = 1$. The computed flow is a flow from node 3 to node n where n is the number of nodes. It is important to notice that this algorithm only works with maximal planar graphs as input since it uses the function dual_rotate_arc which is only working for maximal planar graphs. Further this algorithm produces two files maxGraph.graph and max.dot where maxGraph.graph is the maximal graph obtained at the end of the algorithm in the in the form such that the graph classes can read it and max.dot is the graph in dot language.

6.4 Executables

In this section finds a short description of the executables delivered with the code.

- `project n1 n2 it`: Runs the TestGraph algorithm for all graphs with between n1 and n2 nodes with it iterations. Output: maxGraph.graph, max.dot and stat.dat.
- `project file it`: Runs TestGraph with initial graph from file and with it iterations. Output: maxGraph.graph, max.dot and stat.dat.
- `project`: As the previous function but the parameters are asked by a terminal input.
- `maxflow file s t`: Computes the maximal s-t-flow for the given graph in the file and prints it with intermediate steps to the terminal.

Attention: The input for the files will not be checked by the program. If the given data type does not correspond with the wanted type, the program will crash.

7 Results

In this chapter we analyse the results of the implementations of MaxFlow and TestGraph. So we present different observations. In particular we will have a look on the graphs retured by TestGraph. This graphs should be the graphs that have the longest running time in the MaxFlow algorithm. We will have a look at running time observed in simulations as well as the convergence of the TestGraph algorithm to the graph with highest running time with respect to the number of iterations.

7.1 Graphs

In this section we present some graphs which have been returned form TestGraph algorithm (Algorithm 5).

The arcs in the graphs have an annotation of the form x/y, where x corresponds to the flow over the arc and y corresponds to the capacity of the arc. We recall also that the flow computed is an 3-n-flow where n is the number of nodes ($n = 5, 10$). Since graph visualizations programs like Graphviz do not support planar graph drawing, the biggest graph with 20 nodes will not be presented planar form. The other two graphs are drawn according to the embedding returned by the algorithm. The face f_∞ is the outer face.

The first presented graph is a maximal planar graph with 5 nodes in Figure 2. The value of the flow is 676. The number of needed pushes to obtain this flow is 34. The file describing the graph is in Appendix A.1. This graph is not containing any directed cycles. Further there is a path from node 3 to every other node as well as there is an arc from every node to node 5 and hence also a paht.

MaxFlow algorithm the 5 node planar maximal graph We will present how the MaxFlow algorithm works with help of the graph presented before (Figure 2). Since there are no cycles in the graph, LeftMostCirculation does not add any flow to the initial graph with $f = 0$. Then the algorithm saturates the following path in this order:

- $3 \to 4 \to 1 \to 5$
- $3 \to 4 \to 2 \to 1 \to 5$
- $3 \to 4 \to 2 \to 5$
- $3 \to 4 \to 5$
- $3 \to 5$

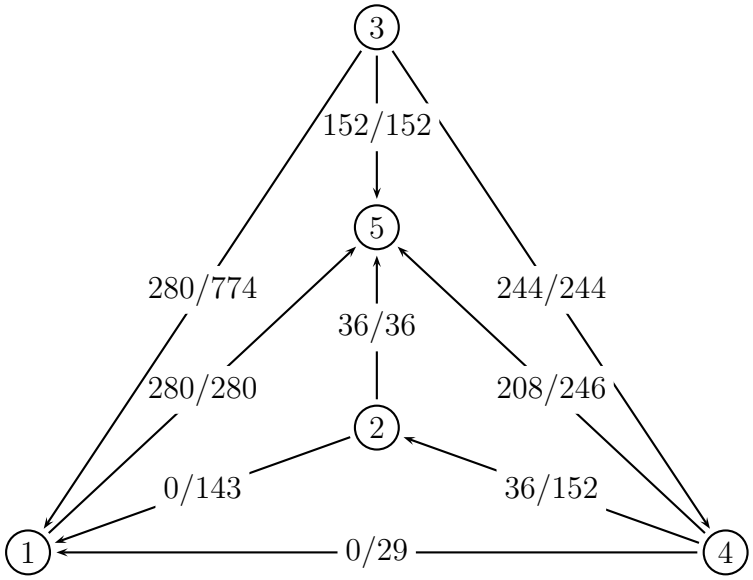


Figure 2: Graph with 5 nodes

- $3 \to 1 \to 5$
- $3 \to 1 \to 2 \to 5$
- $3 \to 1 \to 2 \to 4 \to 5$
- $3 \to 1 \to 4 \to 5$

Those path are in this order always left most in the actual residual graphs which are not displayed here. One observe that as well that the algorithm pushes 4 times over the arc $(3, 4)$ as well over the arc $(3, 1)$ and that over every arc there is at least push.

Next we consider a graph with 10 nodes, which is shown in Figure 3. This graph has 24 arcs. The value of the obtained flow is 5317 and to obtain the graph the algorithm perform 160 pushes. The file describing the graph is in Appendix A.2. Due to the facts discussed in Section 7.2, we can not be totally sure that this is the graph with absolute highest running time but observations show that it is really closed to. We have not seen a graph with 10 nodes with higher running time.

As in the graph with 5 nodes there is no incoming arc for the source node, node 3, and no outgoing nodes for the sink node, node 10. Further there is an arc between node 3 and node 10. There are 3 cycles in this graph which are $6 \to 8 \to 4 \to 1 \to 6$, $6 \to 8 \to 4 \to 9 \to 1 \to 6$ and $6 \to 8 \to 4 \to 5 \to 6$. In LeftMostCirculation those

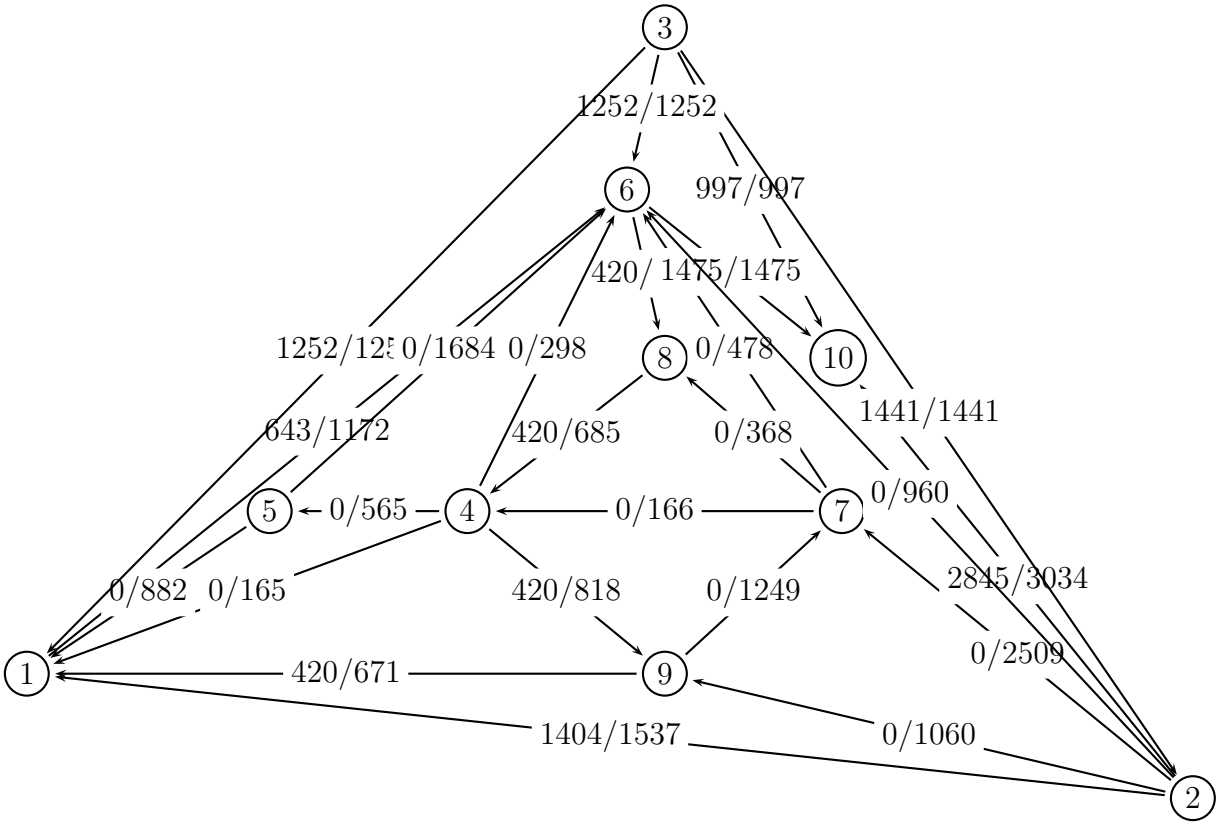


Figure 3: Graph with 10 nodes

circulations are going to be blocked by saturating the first one since this saturates the arc $6 \to 8$ which appears in every circulation.

Every node is reachable from the source node and the sink node is reachable from every node in the graph.

After there are 24 s-t-paths which are going to be saturated in the order followed by LeftMostFlow. The arcs with the most pushes are the arc $3 \to 2$ (11 pushes) and $3 \to 6$ (10 pushes) which are in the set of outgoing arcs of the source node as well as the arcs $6 \to 10$ (10 pushes) and $2 \to 10$ (12 pushes) of the set of incoming arcs to the sink node . Hence there the arcs incident to the source respectively sink are augmented often which is not surprising but this leads to an high number of total performed pushes.

If one compares the graph with 5 nodes and the graph with 10 nodes one observe that there is in both graphs a direct arc from source to sink, this appears surprising in the first moment but this allows to have longer other s-t paths and hence a higher number of total pushes and running time which is maximized in those two graphs. It is also important to see that there are no incoming respectively outgoing arcs in the sink respectively source

node. The graphs are created such that there are s-t-paths which are close the the possible maximum of $n-1$ arcs.

Since every node is reachable from the source node and it is possible to reach the sink node from every node this leads to a hight number of possible paths and the fact that there are paths where first there is flow pushed in one direction and after in the opposite direction, this increases the number of possible paths. Further with the found structure LeftMostFlow augments several times over the same arc. With the specific version of the LeftMostFlow algorithm this is minimized which leads afterwards to a smaller number of pushes.

One may observe that the usability theorem is not violated by performing the algorithm,

7.2 Convergence of TestGraph

In this section we discuss the convergence or the test graph algorithm with help of the output of a simulation of TestGraph algorithm for a graph with 5 nodes and a graph with 10 nodes.

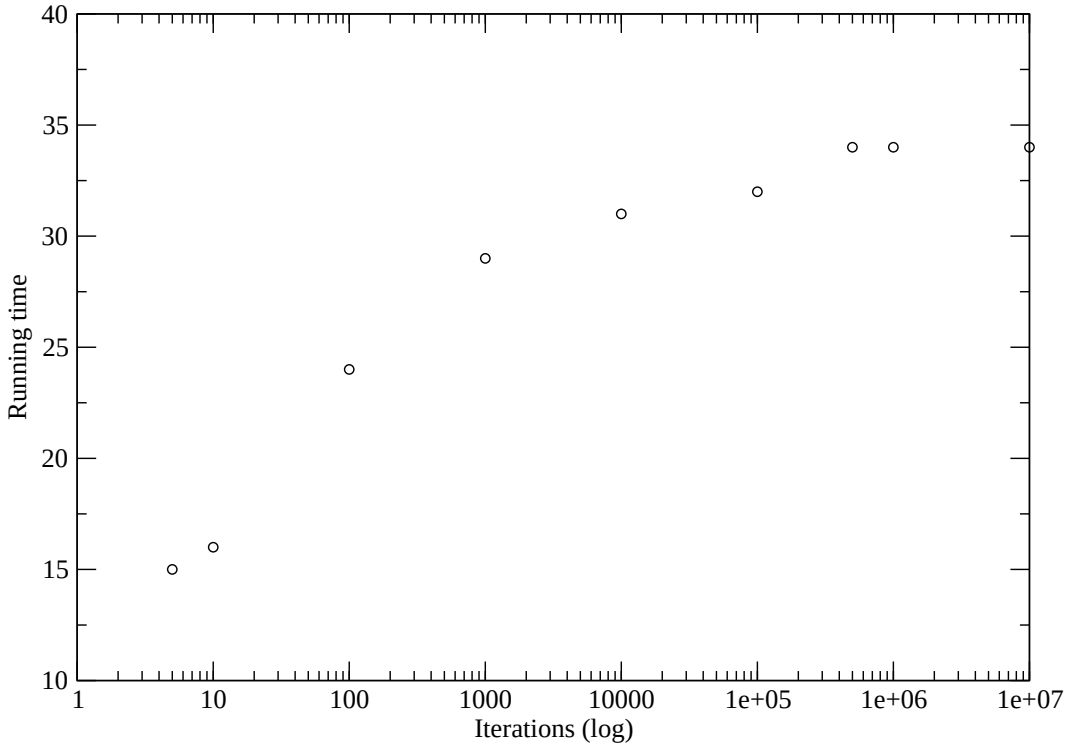


Figure 4: Convergence of the running time in the number of iterations for a graph with 5 nodes

Let us consider first the result fo the graph with 5 nodes which is presented in Figure 4. One observe that he convergence is quite slow. Note that the scale of the x-axis is logarithmic. The slope of the convergence curve gets flat around 10'0000 iterations. So an approximate maximum is reached after 10'000 iterations since there is no noise it seems that around this value the real maximum for the running time is reached.

As we consider a graph with 10 nodes we get Figure 5 where one may observe that it is less obvious to tell if we reached the maximum. At 10'000'000 iterations the maximum

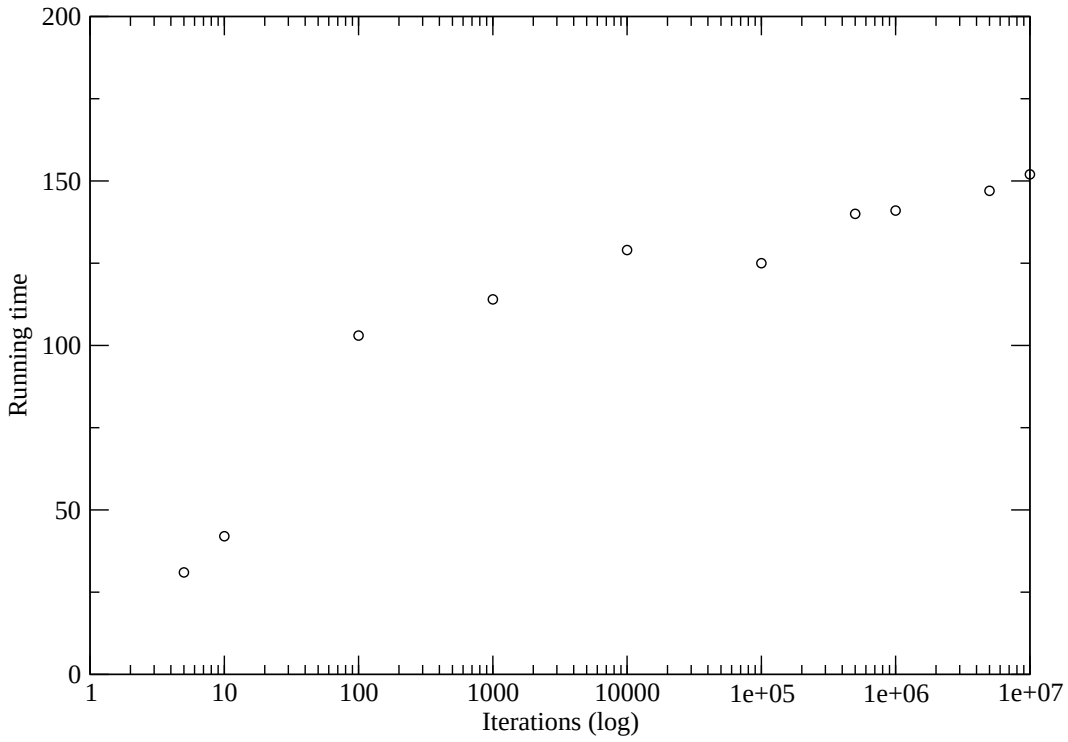


Figure 5: Convergence of the running time in the number of iterations for a graph with 10 nodes

is not reached yet. The running time does increase slowly as it can be seen in the log-lin plot. One may also observe effect of the randomized algorithm by the noise of the different observations.

By the observations of the development of the maximal running time by increasing number of iterations for the graphs with 5 and 10 nodes, we deduce that a high number of iterations is needed in order to obtain an approximation of the maximal running time with the algorithm TestGraph. Further one may also observe that the result is influenced by the fact that the algorithm is randomized.

The number of iterations needed to obtain an approximate maximal running time is increasing at least at factor 100 comparing a graph with 5 nodes with a graph with 10 nodes. Hence the number of needed iterations is increasing in a enorm number for an increasing number of nodes and the convergence of TestGraph gets worse and worse. Its possible that this bad behaviour comes from the fact that some graphs are visited several times.

It would hence be interesting and useful to analyse the TestGraph algorithm better and improve its convergence for further investigations.

7.3 Simulated maximal running time for MaxFlow

In this section we investigate the simulated maximal running time of our implementation of the MaxFlow algorithm and compare it to the theoretical bound of Theorem 7 which is $O(n \log n)$. The data used are the returned values of the TestGraph algorithm. Note that the number of iterations is 20'000'000. This works for the smaller number of nodes but for bigger number of nodes this might not be sufficient following the discussion of Section 7.2. The results are displayed in Figure 6 and are available in Appendix B. First one may

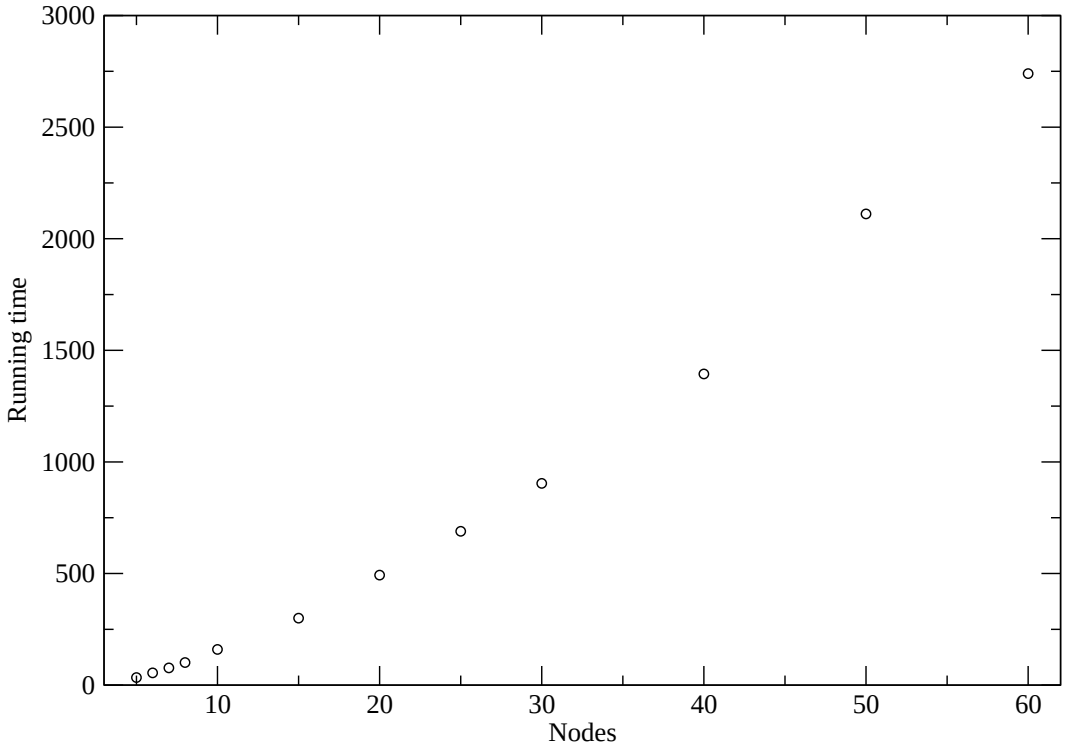


Figure 6: Simulated maximal runtime

observe that the acceptance rate is between 50% and 60%. However changes of β did not have leed to big changes for the acceptance rate. This implies that most of the time the arc flips lead already to higher running time. Which is corresponding to the slow convergence since this implies that the changes are rather small. The curve of the different graphs is clearly not linear. We will now consider some fitting curves.

As our hypothesis is that the running time is $O(n \log n)$ we consider first a linear function in $n \log n$. This function can be obtained by an regression analysis. The obtained function is

$$y = 11.17 \cdot x \log x - 124.23. \tag{1}$$

The highly negative second term indicates that this function is probably not good for the small values.

In Figure 7 we can compare the linear function with the obtained data in a log-log plot. As expected the curve fits bad for small values but also for the biggest value, the value of

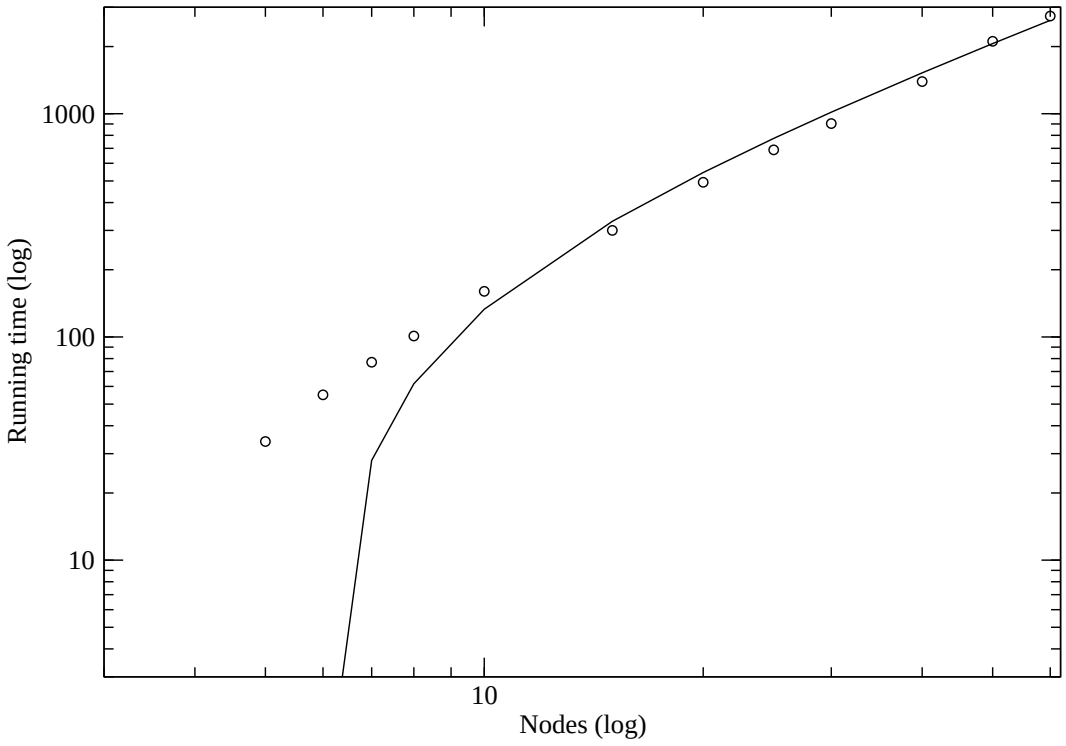


Figure 7: $n \log n$-linear approximation (Eq.(1))

the function (1) is smaller than the value obtained in the simulation. But following the discussion in the previous section this value is smaller than the real value. And so the function linear in $n \log n$ is not as good function to describe the data.

Since function (1) fits not good the data, we continue to find a fitting function. Since we rejected the linear hypothesis we consider now a power function. The power function which fits the data best is

$$y = 2.69 \cdot x^{1.71}. \tag{2}$$

The power is in between 1 and 2 and hence if this function fits this algorithm is better than a $O(n^2)$ algorithm since its $O(n^{1.71})$. In Figure 8 the power function is drawn in a lin-lin and a log-log plot with the data in order to compare it to the data obtained in the simulation. The power function (2) fits the data well and it is, unlike the $n \log n$ linear

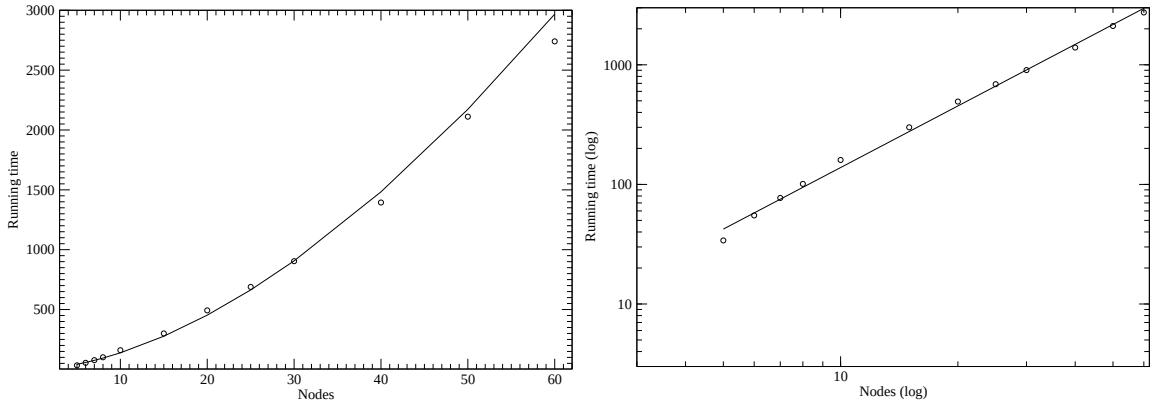


Figure 8: n-power approximation (Eq.(2))

function bigger than the data for the graph with 60 nodes, which is to be more likely. It follows that the data obtained by the simulation is well described by the power function (2). Hence our MaxFlow algorithm has approximately a maximal running time of $O(n^{1.71})$. And it is hence not linear in $n \log n$.

This is in contradiction with what we expected from Theorem 7, but it is not contradicting it. Since we didn't implement the specific version of the LeftMostFlow algorithm (Algorithm 4) but the general Algorithm 3. Hence one must implement the specific LeftMostFlow algorithm to obtain an real $O(n \log n)$ algorithm. But still an maximum flow algorithm in $O(n^{1.71})$ is quite good and the not specific implementation would allow on a easier way to improve the algorithm.

8 Conclusion

We presented the maximum flow algorithm of Borradaile and Klein as well as the algorithm to find the maximal planar graph with highest running time. There is also an implementation of those to algorithm in C++ which we discussed in Section 6.

We implemented not the specific version of the LeftMostFlow which leads to an $O(n \log n$ algorithm but only the generic version which leads to an longer running time but it is less strict and easier to improve at the moment an new idea arises.

We observed that the implemented maximum flow algorithm has in the worst case a running time which is approximately $O(n^{1.71})$ which is still decent but not $O(n \log n)$. This algorithm will improve if instead of the actual used, with an error, algorithm one uses the specific algorithm.

For the TestGraph algorithm we observed that the convergence is slow and hence it needs high number of iterations compared to the number of nodes to obtain worst running time graph. This has some drawbacks to the interpretations of the maximal running time graphs since one may not be sure that it is, specially for a big number of nodes, the graph with the maximal running time or just a graph which is close to. We also observed that the acceptance rate is between 50% and 60%.

We also made some observation about the graphs which have maximal running time and there one could see why the specific version of the LeftMostFlow version is needed to obtain an $O(n \log n)$ algorithm

Overall we got a decent maximum flow algorithm for directed planar graphs and a algorithm to find the maximal running time which could even with small modifications be used to approximate any other searched value, but at least with respect to the implemented maximum flow algorithm the convergence is slow and it would be interesting to improve this property.

References

[1] Glencora Borradaile and Philip Klein. An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. *J. ACM*, 56(2):Art. 9, 30, 2009.

[2] G. Brinkmann and Brendan D. McKay. Construction of planar triangulations with minimum degree 5. *Discrete Math.*, 301(2-3):147–163, 2005.

[3] Dominique de Werra, Thomas M. Liebling, and Jean-Franois Hêche. *Recherche oprationnelle pour ingnieurs I*. Enseignement des mathmatiques. PPUR, Lausanne, 2003.

[4] Stefan Felsner. *Geometric graphs and arrangements*. Advanced Lectures in Mathematics. Friedr. Vieweg & Sohn, Wiesbaden, 2004. Some chapters from combinatorial geometry.

[5] Samir Khuller, Joseph Naor, and Philip Klein. The lattice structure of flow in planar graphs. *SIAM J. Discrete Math.*, 6(3):477–490, 1993.

[6] Charles L. Lawson. Transforming triangulations. *Discrete Math.*, 3:365–372, 1972.

[7] Gary L. Miller and Joseph Naor. Flow in planar graphs with multiple sources and sinks. *SIAM J. Comput.*, 24(5):1002–1017, 1995.

[8] Alexander Schrijver. On the history of the transportation and maximum flow problems. *Math. Program.*, 91(3, Ser. B):437–445, 2002. ISMP 2000, Part 1 (Atlanta, GA).

[9] Alexander Schrijver. *Combinatorial optimization. Polyhedra and efficiency. Vol. A*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 2003. Paths, flows, matchings, Chapters 1–38.

A Graphs

A.1 Graph with 5 nodes

```
5  9
1  3  −8
1  4  −7
1  2  −4
1  5  2
2  4  −5
2  5  3
2  1  4
3  1  8
3  5  6
3  4  1
4  1  7
4  3  −1
4  5  9
4  2  5
5  4  −9
5  3  −6
5  1  −2
5  2  −3
Capacities and flows∗
1 244 244
2 280 280
3 36 36
4 143 0
5 152 36
6 152 152
7 29 0
8 774 280
9 246 208

34
```

A.2 Graph with 10 nodes

```
10  24
1  6  12
1  3  −21
1  2  4
1  9  −22
```

1	4	−18
1	5	−20
2	10	6
2	6	23
2	7	2
2	9	15
2	1	−4
2	3	−13
3	1	21
3	6	19
3	10	3
3	2	13
4	6	10
4	5	7
4	1	18
4	9	11
4	7	−17
4	8	−8
5	4	−7
5	6	24
5	1	20
6	3	−19
6	1	−12
6	5	−24
6	4	−10
6	8	1
6	7	−5
6	2	−23
6	10	9
7	4	17
7	9	−14
7	2	−2
7	6	5
7	8	16
8	6	−1
8	4	8
8	7	−16
9	1	22
9	2	−15
9	7	14
9	4	−11
10	2	−6
10	3	−3

```
10  6  -9
Capacities and flows*
1 420 420
2 2509 0
3 997 997
4 1537 1404
5 478 0
6 3034 2845
7 565 0
8 685 420
9 1475 1475
10 298 0
11 818 420
12 1172 643
13 1441 1441
14 1248 0
15 1060 0
16 368 0
17 166 0
18 165 0
19 1252 1252
20 882 0
21 1627 1627
22 671 420
23 960 0
24 1684 0

160
```

B Running time data

```
5  34  20000000 11925828 15.2477
6  55  20000000 11657198 21.9116
7  77  20000000 11458987 29.158
8  101  20000000 11315512 36.9968
10  160  20000000 11081146 54.625
15  300  20000000 10985310 105.227
20  493  20000000 10922611 166.948
25  689  20000000 10909792 238.251
30  904  20000000 10884723 319.587
40  1394  20000000 10891723 506.74
50  2111  20000000 10924491 723.933
60  2740  20000000 10917509 979.505
```