Maximum flow in a network

Technical Analysis

Piotr Janaszek

Faculty of Mathematics and Information Science
Warsaw University of Technology

29 March 2012

Document Metric

Project:	Maximum flow in a network		Company:	WUT	
Name:	Technical Analysis				
Topics:	Technical analysis, system architecture, used algorithms				
Author:	Piotr Janaszek				
File:	TechnicalAnalysis.pdf				
Version no:	1.1	Status:	Final	Opening Date	09.03.2012
Summary:	Brief description of technical base of the application				
Authorized by:				Last Modification Date:	24.03.2012

History of changes

Version	Date	Author	Description
0.1	09.03.2012	Piotr Janaszek	Created document
0.2	10.03.2012	Piotr Janaszek	Created Preface For Management
0.4	13.03.2012	Piotr Janaszek	Created Application Overview
0.5	16.02.2012	Piotr Janaszek	Added Used Solutions and System Architecture
0.7	17.03.2012	Piotr Janaszek	Added Used Algorithms
0.9	20.03.2012	Piotr Janaszek	Created User Interface and Post face
1.0	21.03.2012	Piotr Janaszek	Fixed mistakes and changed document formatting
1.1	24.03.2012	Piotr Janaszek	Updated User Interface section, updated activity diagram

TABLE OF CONTENTS

PREFACE FOR MANAGEMENT

This document is presenting the technical aspects of the application. It is said to present analysis of used solutions and provided functionalities.

The aim of the project is to develop the application which calculates maximum flow in a network. The analysis should be done using Ford-Fulkerson, Edmonds-Karp and Dinitz blocking algorithms. The application is said to visualize input graph and calculate maximum flow depending on values specified by the user.

The problem of maximum flow is common in computer science but is also present in other areas of life such as transportation. Hence the application should ease the calculations in a convenient way.

The application is supposed to be user-friendly. Hence the user interface should be clear and intuitive in use. Also system architecture should be designed to give maximum reliability for the user needs and future update purposes.

In the *Application Overview* section there are main functionalities of the application described and simple activity graph shown.

The following sections are going deeper and are showing the reader the application from the inside, mainly used technologies, algorithms and solutions.

APPLICATION OVERVIEW

In this section a reader can find short and brief description of the application from a distance without going deep into the technical details of the program. It will help the end-user to easily understand next sections of this document and get to know how the whole application is working.

Main functionalities

The application main functionality consists of several simple tasks:

1. Get data (either from opened file or manually via GUI).
2. Create graph structure.
3. Calculate maximum flow.
4. Plot the resulting graph.
5. Display summary and show it to the user (it may be saved to external file).
6. Export graph structure to external file.

The functionalities listed above are the most important parts in the application. All other features will come directly from them or will be added in order to develop the useful and reliable program.

Application activity

Almost all tasks given to the program are said to be done sequentially, because of a step-by step manner of the application. One action can be taken only if the previous one is finished.

The easy and convenient way of showing the scheme of activities is the activity diagram. The figure below shows the activity of the application in time. It is clearly visible that retrieving data is done in a loop until no more data will be considered during the execution. Calculating the maximum flow is done right before preparing the results.

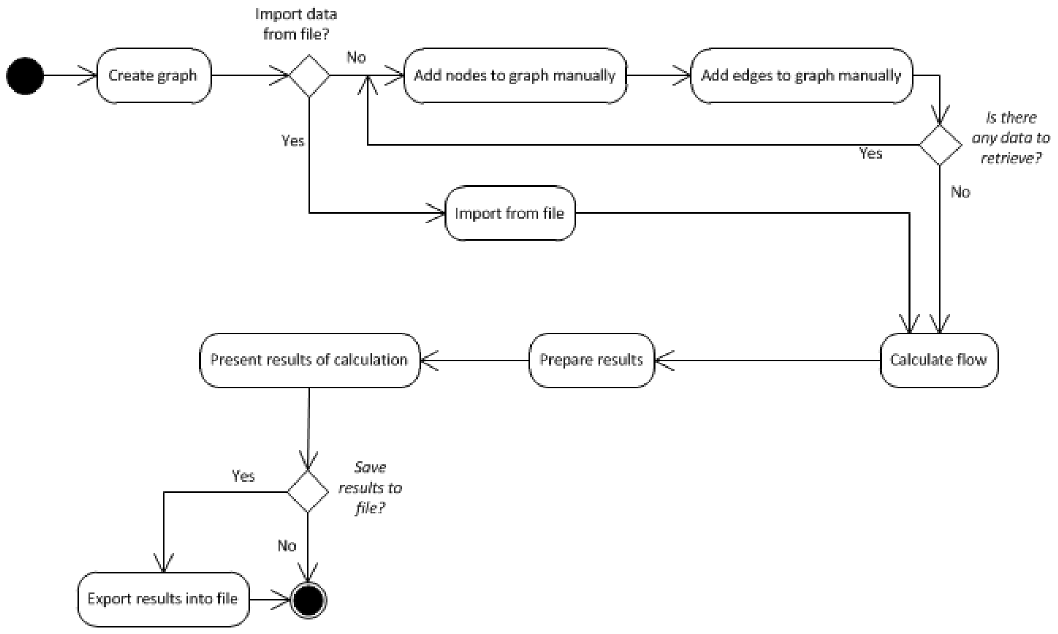


Figure 1: Activity diagram of the application

USED SOLUTIONS

In this section the data files used by the application is described in details as well as environment used to develop the program.

External data

Both input and output files used for importing the graph and exporting current graph are in XML format. The general structure of the file tree is as in the following example:

```xml
<?xml version="1.0" encoding="utf-8"?>
<graph>
  <vertices>
    <vertex>
      <id>vertex1</id>
      <mode>source</mode>
    </vertex>
    <vertex>
      <id>vertex2</id>
      <mode>normal</mode>
    </vertex>
    <vertex>
      <id>vertex3</id>
      <mode>sink</mode>
    </vertex>
  </vertices>
  <edges>
    <edge>
      <from>vertex1</from>
      <to>vertex2</to>
      <flow>5</flow>
    </edge>
    <edge>
      <from>vertex3</from>
      <to>vertex2</to>
      <flow>10</flow>
    </edge>
  </edges>
</graph>
```

The vertex mode element can be empty. Then it means that the given vertex is “normal” vertex i.e. it is not a source neither a sink vertex. It may happen that the file will be bad-formatted. For example there will be several source vertices and several sink vertices. The application while parsing will set the last occurrence of this special vertex type and set all previous ones to normal state. All data outside the graph element is ignored.

Results file

The results file which can be saved after maximum flow is calculated is in XML format. It gives information about the utilization of each edge’s capacity and maximum value. It is very similar to the input file but contains one additional element in edge section mainly `<used_capacity></used_capacity>`.

Environment

The application is based on Microsoft solutions. Technology upon which it will developed is Windows Forms and C# programming language. Chosen Integrated Development Environment is Microsoft Visual Studio 2010 since it provides all necessary features and functionality for creating modern and reliable applications and unified solutions.

USED ALGORITHMS

For proper processing of the input file in the logic layer of the system architecture, there are some algorithms needed.

Ford–Fulkerson Algorithm

The Ford–Fulkerson Method is an algorithm which computes the maximum flow in a flow network. The idea behind the algorithm is very simple. As long as there is a path from the source (start node) to the sink (end node), with available capacity on all edges in the path, we send flow along one of these paths. Then we find another path, and so on. A path with available capacity is called an augmenting path. The way in which we search for a path with excess capacity is not specified. Thus this algorithm in specific situations may not terminate or may produce wrong results.

Pseudo code[1]:

```
Input: graph $G$ with flow capacity $c$, a source node $s$, and a sink node $t$
Output: a flow $f$ from $s$ to $t$ which is a maximum
    1. $f(u,v) \leftarrow 0$ for all edges $(u,v)$
    2. While there is a path $p$ from $s$ to $t$ in $G_f$ such that $c_f(u,v) > 0$ for all edges $(u,v) \in p$:
        1. Find all $c_f(p) = \min\{c_f(u,v) : (u,v) \in p\}$
        2. For each edge $(u,v) \in p$
            1. $f(u,v) \leftarrow f(u,v) + c_f(p)$ (Send flow along the path)
            2. $f(v,u) \leftarrow f(v,u) + c_f(p)$ (The flow will be returned later)
```

When no more paths in step 2 can be found, s will not be able to reach t in the residual network. If S is the set of nodes reachable by s in the residual network, then the total capacity in the original network of edges from S to the remainder V is on the one hand equal to the total flow we found from s to t, and on the other hand serves as an upper bound for all such flows. This proves that the flow we found is maximal.

Complexity:

When the capacities are integers, the time of Ford-Fulkerson is bounded by $O(Ef)$, where E is the number of edges and f is the maximum flow in the graph.

Edmonds-Karp Algorithm

The algorithm is identical to the Ford–Fulkerson algorithm, except that the search order when finding the augmenting path is defined (breadth-first search). The path found must be a shortest path that has available capacity. This can be found by a breadth-first search, as we let edges have unit length. The running time of $O(VE^2)$ is found by showing that each augmenting path can be found in $O(E)$ time, that every time at least one of the E edges becomes saturated, that the distance

[1] Source: http://en.wikipedia.org/wiki/Ford%E2%80%93Fulkerson_algorithm

from the saturated edge to the source along the augmenting path must be longer than last time it was saturated, and that the length is at most V. Another property of this algorithm is that the length of the shortest augmenting path increases monotonically[2].

Dinitz Blocking Algorithm

Dinic's algorithm is a strongly polynomial algorithm for computing the maximum flow in a flow network. The algorithm runs in$O(V^2E)$ time and is similar to the Edmonds–Karp algorithm, which runs in $O(VE^2)$ time, in that it uses shortest augmenting paths. The introduction of the concepts of the level graph and blocking flow enable Dinic's algorithm to achieve its performance.

Pseudo code[3]:

```
Input: A network $G = \left((V, E), c, s, t\right)$
Output: A $s - t$ flow $f$ maximum value
    1. Set $f(e) = 0$ for each $e \in E$
    2. Construct $G_L$ from $G_f$ of $G$. If $dist(t) = \infty$, stop and output $f$
    3. Find a blocking $f'$ in $G_L$
    4. Augment flow $f$ by $f'$ and go back to step 2
```

[2] Source: http://en.wikipedia.org/wiki/Edmonds%E2%80%93Karp_algorithm
[3] Source: http://en.wikipedia.org/wiki/Dinic%27s_algorithm

SYSTEM ARCHITECTURE

System architecture consists of two parts: presentation and logic layer. The former is responsible for communication between the application and the user. The latter is processing the signal and creating the output.

Presentation Layer

This layer as mentioned above is responsible for communication between the application and the user. In practice, it means that the program should handle the information transferred from user to the application and vice versa. Due to that the user interface including:

- Buttons for launching methods,
- combo boxes for switching the options,
- image box for plotting and
- labels for presenting information

must lie in the presentation layer. But all the functionality triggered by using it should lie in another layer, mainly logic layer. These assumptions bring the reliability to the interface. But also the program should have an intuitive interface for better communication. The reader can read more about that part in *User Interface* section on page 12.

Logic Layer

Logic layer is responsible for covering all the functionality that is not visible to the user. It is very important for the project, because the presentation of the results even in the most pretty way makes no sense when the results are incorrect.

For example, the input file must be loaded correctly with no errors. It must be loaded from given directory, in proper format, as one whole piece and without making any changes in the content.

Logic layer must be divided from the presentation layer for better reliability. In this project all application logic is done by the dynamically loaded external library called *Provider*.

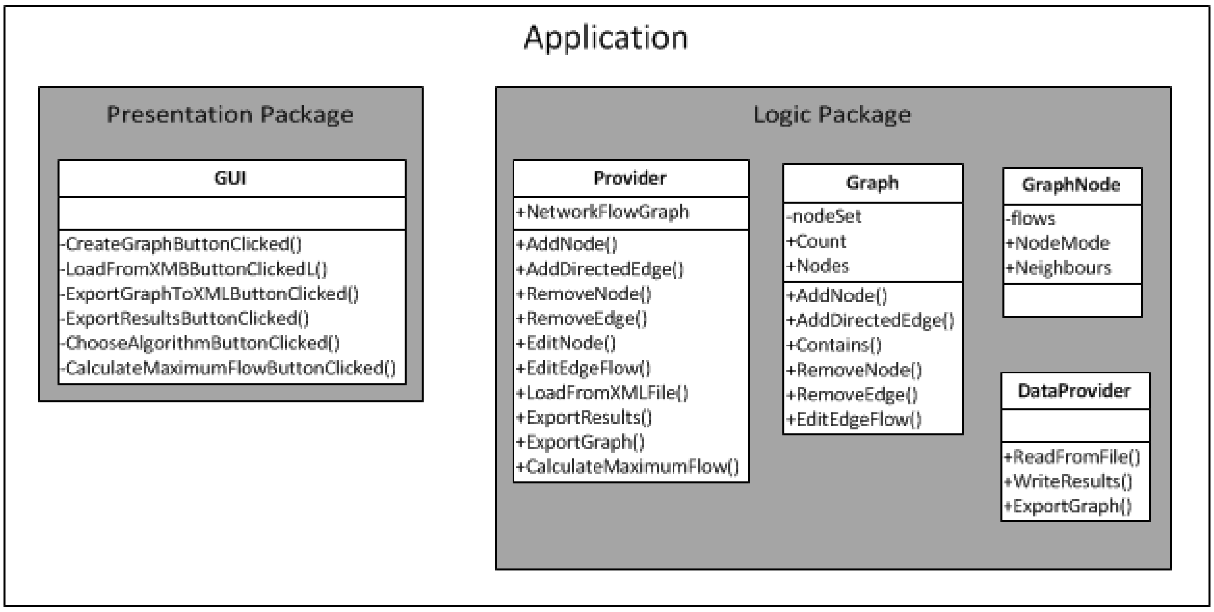


Figure 2: Class diagram of the application

Graph class is responsible for holding fields and methods allowing low level graph manipulation such as add, remove node or edge. DataProvider class is responsible for reading and writing files to the disk such as importing and exporting graphs. The provider class is general API which is exposed to GUI package. It controls all actions done on graph via user interface, manipulates the graph and returns the results.

USER INTERFACE

User interface consists of one main window which is built from several components. Mainly menu bar (File, Edit, Process, About), tool strip bar, where all often used functions are represented graphically and area to visualize the graph. Below there is also a status bar where the user can find some information about current selected object under mouse or progress of calculation.

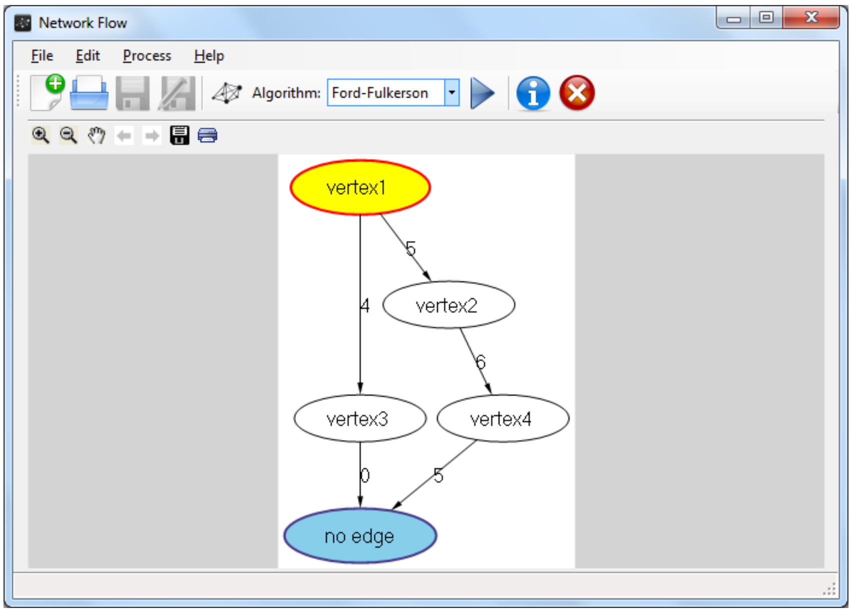


Figure 3: Main window layout mockup

All activities are done using point-and-click technique, hence the program is easy to use. For each action there appears a new dialog box where all necessary fields must be specified. This highly increases the readability of the application.

Graph Visualization

Graphical representation of a network is done with use of Microsoft GLEE library. This component have methods implemented allowing to zoom, scroll, print and export graph to an image file. Nodes and edges layout is done automatically and the user is not able to change it manually. Each time a node or an edge is added or removed the graph layout is recalculated in such a way that the entire graph is visible at once. It has one main disadvantage i.e. if we put a lot of vertices with edges the automatic algorithm will zoom out everything in order to match the screen.

Source and sink nodes have special representation in order to be distinguishable from other regular nodes. And source has red line with yellow filling whereas sink has blue line with dark blue filling.

There exists also “Step by step” mode which can be enabled under Process drop down list. When on each time the calculate button is pressed only one cycle of calculation is performed and presented to the user.

Graph Manipulation

Graph manipulation is done, as mentioned before, with use of point and click technique. The user can add, edit or remove nodes and edges via context menu placed on the visualization control. After clicking on each item a new dialog box is displayed. The following screen shots demonstrates each type of possible activity.

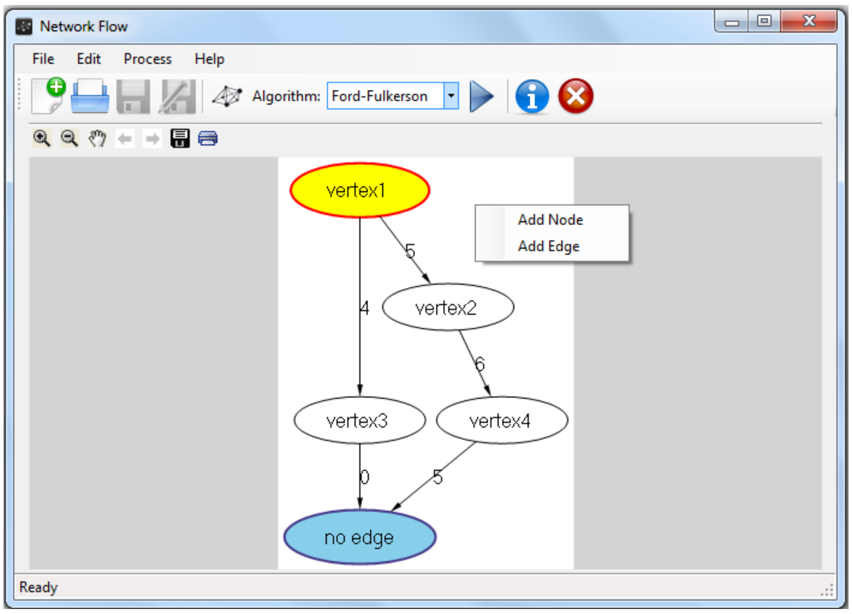


Figure 4: Context menu when no object is under mouse

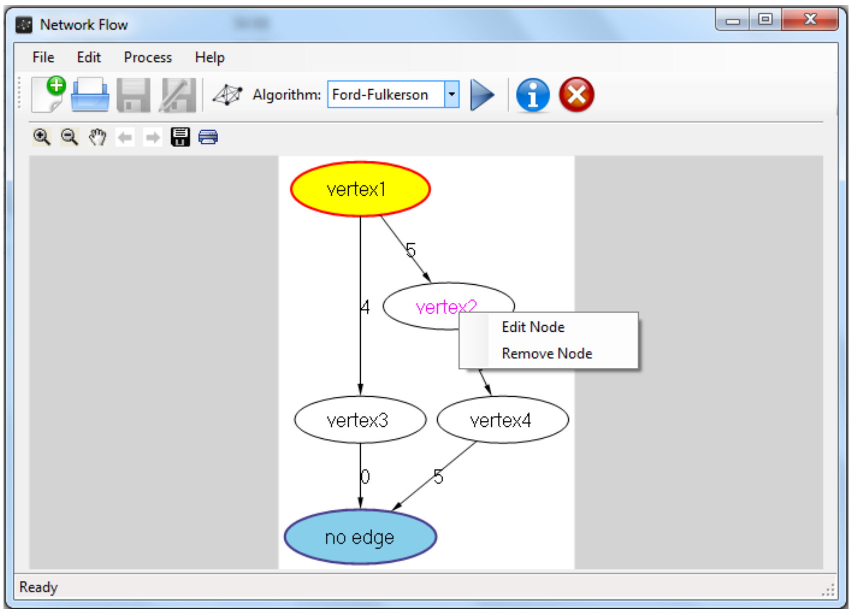


Figure 5: Context menu when node object is under mouse

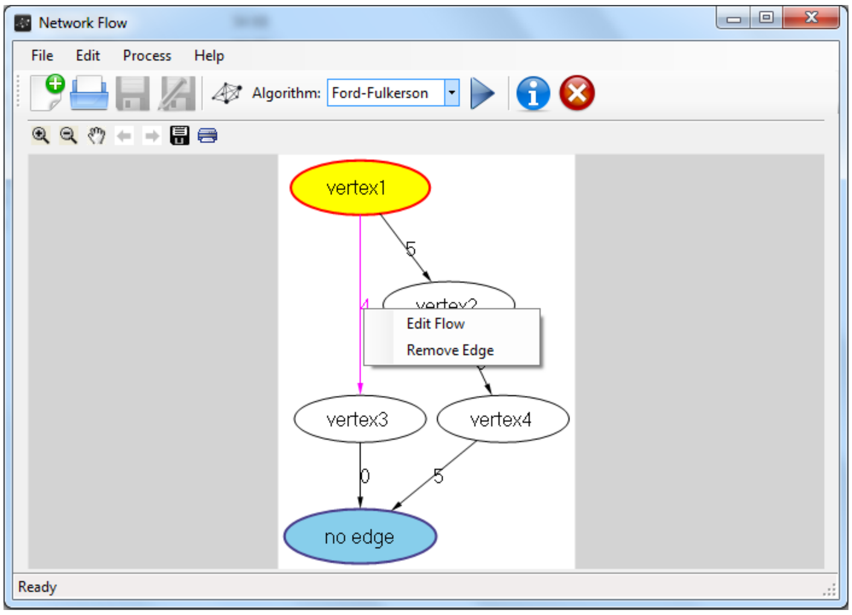
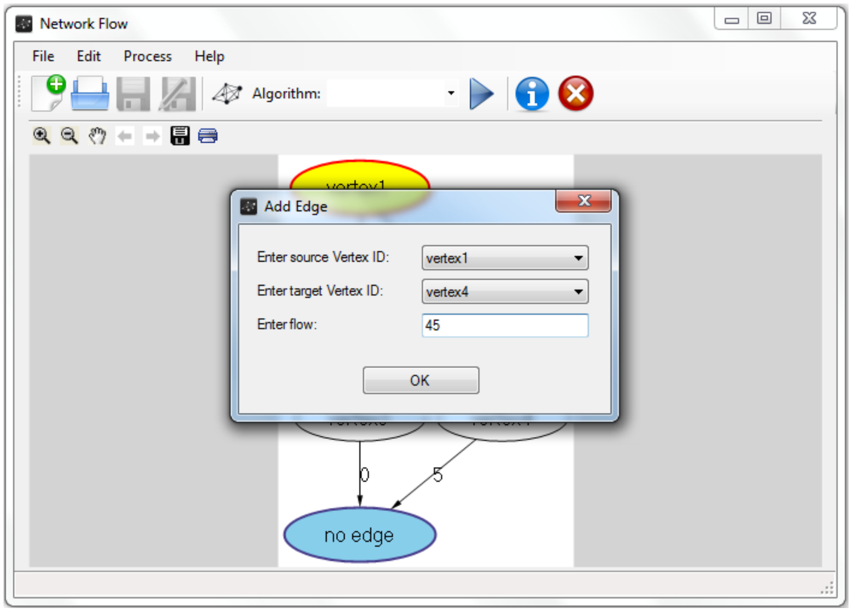


Figure 6: Context menu when edge object is under mouse



Enter source Vertex ID: vertex1
Enter target Vertex ID: vertex4
Enter flow: 45

Figure 7: Dialog box example

POST FACE

The application functionality is rather simple and it is designed to calculate maximal flow in a network, create the graph diagram and display the results which can be saved into the hard disk. It uses well-known Ford-Fulkerson, Edmonds-Karp and Dinitz Blocking algorithms.