# The First DIMACS International Algorithm Implementation Challenge: Problem Definitions and Specifications

January 3, 2016

## 1 Introduction

The Center for Discrete Mathematics and Theoretical Computer Science (DIMACS) invites participation in an international Implementation Challenge to find and evaluate efficient and robust implementations of algorithms for the *minimum-cost flow* problem, the *maximum flow* problem, the *assignment* problem, and the *non-bipartite matching* problem.

Participants are invited to carry out research projects related to these problems and to present research papers at a DIMACS workshop to be held in Fall of 1991. To obtain more information, send an email request for the document *General Information* to **netflow@dimacs.rutgers.edu**. Request either a LaTeX version (sent through email) or a hard copy (sent through U.S. Mail), and include return address as appropriate.

Projects may involve implementing and testing algorithms, building input generators, or developing implementations for newer architectures. DIMACS will provide a clearinghouse for exchange of input generators, programming support tools, guidelines for experimental research, and a set of benchmarks for each problem.

To facilitate exchange of generators and implementations, standard problem definitions and input/output formats are presented in this document. There is no requirement that participants follow these specifications; however, compatible implementations will be able to make full use of the DIMACS support tools. Depending upon the projects chosen by participants, new input formats may be developed for problem areas of special interest.

Several texts describe algorithms for computing network flows: see for example

Derigs [?], Jensen and Barnes [?], Lawler [?], Kennington and Helgason [?], Papadimitriou and Steiglitz [?], or Tarjan [?]. Two excellent surveys are those by Ahuja et al. [?] and Goldberg et al. [?]. A recent bibliography by Veldhorst [?] contains nearly 300 references. Matching problems are described in Avis [?], Lawler [?], Lovász and Plummer [?], Papadimitriou and Steiglitz [?] and Tarjan [?]. Many new algorithms for these and related problems may be found in recent STOC, FOCS, and SODA proceedings; see also journals such as *Journal of Algorithms*, *Journal of the ACM*, *Management Science*, *Mathematical Programming*, *Operations Research*, and *SIAM Journal on Computing*.

Section 2 gives general specifications for the three network problems (minimum-cost flow, maximum flow, and assignment). Problem definitions and input formats specific to each network problem are given in Sections 3, 4, and 5. Section 6 presents problem definitions and formats for the matching problem.

## 2 Networks and Flows

### 2.1 Network Definitions

A network is a set $G = (N, A)$ of $n$ nodes and $m$ directed arcs. Associated with each arc $(v, w) \in A$ is a lower bound, $low(v, w) \geq 0$ and a capacity, $cap(v, w) \geq low(v, w)$. Each arc $(v, w)$ also has a cost, $cost(v, w)$. Associated with each node $v$ is a node flow value $b(v)$. Node $v$ is called a supply node, a demand node, or a transshipment node depending upon whether $b(v)$ is greater than, less than, or equal to zero, respectively.

A flow assignment, $f(v, w)$, defined on arcs in $G$, has the following properties.

- Arc capacity constraints are maintained: $low(v, w) \leq flow(v, w) \leq cap(v, w)$ for all $(v, w) \in A$.
- Flow is conserved at all nodes: $\sum_u f(u, v) - \sum_w f(v, w) = b(v)$ for all $v \in N$.

### 2.2 Network Structure.

The following assumptions will be made about networks provided as inputs for the Implementation Challenge.

- For a network with $n$ nodes it is assumed that the nodes are identified by the integers $1 \ldots n$.
- All costs, capacities, flows, and other numerical quantities are integer valued.

- For the official benchmark instances, all input values are in the range $[-2^{31} \ldots 2^{31}-1]$. Larger magnitudes are acceptable for other experiments.
- It is assumed that $0 \leq \mathit{low}(v,w) \leq \mathit{cap}(v,w)$ for all $(v,w) \in A$.
- An uncapacitated arc (i.e. $\mathit{cap}(v,w) = \infty$) is identified by a negative integer value for $\mathit{cap}(v,w)$. Programs reading a negative capacity may assign an appropriate MAXINT value or may mark such arcs for special treatment.
- There is no *a priori* restriction on the number of nodes $n$ or the number of arcs $m$.
- Self-arcs are not allowed. That is, there are no arcs of the form $(v,v)$ in the input networks.

## 3 Minimum-Cost Flows

### 3.1 Problem Definition

An input instance for the minimum-cost flow problem is a network with capacities, lower bounds, costs, and node flow values as defined in Section 2. A maximum flow is one in which

$$\sum_{(v,w)\in A} f(v,w)$$

is maximized. The minimum-cost flow problem is to find a maximum flow such that the *total flow cost*

$$\sum_{(v,w)\in A} f(v,w)\cdot \mathit{cost}(v,w)$$

is minimized.

### 3.2 Network Structure.

The following assumptions are made about network structure for the minimum-cost flow problem.

- There may be multiple arcs $(v,w)$ between any pair of nodes $v$ and $w$. The arcs may have differing costs, capacities and lower bounds.
- It is not necessarily the case that $(v,w) \in A$ implies $(w,v) \in A$.

- If both $(v,w)$ and $(w,v)$ are in $A$, it is not necessarily the case that $cost(v,w) = -cost(w,v)$.
- It is *not* assumed that the network has a feasible solution, i.e. that a legal flow assignment exists in the network.
- It is *not* assumed that the network is connected.

## 4 Maximum Flow

### 4.1 Problem Definition

Input instances for the maximum flow problem are restrictions of the general networks defined in Section 2. For this problem the network contains a unique *source* node $s$ and a unique *sink* node $t$. All other node are transshipment nodes (with $b(v) = 0$). The flow conservation property is redefined to hold at transshipment nodes, but not at the source and sink nodes. All arc lower bounds $low(v,w)$ are assumed to be 0 and all arc costs $cost(v,w)$ are assumed to be 1.

The problem is to find a flow assignment $f(v,w)$, defined on arcs in $G$, which maximizes the amount of flow from the source to the sink. That is, the maximum flow is one for which

$$\sum_{v:(s,v)\in A} f(s,v)$$

is maximized, with the condition that no flow is allowed *into* the source (i.e. $\sum(v,s) = 0$) and no flow is allowed *out of* the sink ($\sum(t,v) = 0$).

### 4.2 Network Structure.

The following further assumptions are made about network structure for the maximum flow problem.

- There are no uncapacitated edges (i.e. with $cap(v,w) = \infty$).
- It is not assumed that there exists a path from $s$ to $t$.
- Multiple arcs between any vertex pair $(v,w)$ are not allowed.

# 5 Assignment

## 5.1 Problem Definition

Inputs for the assignment problem are restrictions of the general networks defined in Section 2. Instances for assignment must be *bipartite*: that is, the set of nodes $N$ can be partitioned into two sets $N_1$ and $N_2$ such that for all arcs $(v, w) \in A$, $v \in N_1$ and $w \in N_2$. Node flow values obey $b(v) = 1$ for $v \in N_1$ and $b(w) = -1$ for $w \in N_2$. All arc capacities are 1 and all lower bounds are 0. Arc cost $cost(v, w)$ is as defined for general networks.

The assignment problem is to find the flow assignment $f(v, w)$, defined over arcs in $G$, which maximizes

$$\sum_{(v,w)\in A} f(v, w) \cdot cost(v, w).$$

Note that by the constraints on flows and nodes and by the assumption of integral values, $f(v, w)$ is either 0 or 1 for each arc $(v, w)$. Furthermore, by the flow conservation property no two unit-valued flows share a common vertex. Therefore the flow solution $f(v, w)$ defines a matching which maximizes total cost.

## 5.2 Network Structure.

The following assumptions are made about network structure for the assignment problem.

- The set $N_1$ is a subset of the integers from 1 through $n$. It is assumed that $N_2 = N - N_1$.
- It is *not* assumed that $|N_1| = |N_2|$, or that perfect matching exists in the network.
- Multiple arcs between any vertex pair $(v, w)$ are not allowed.

# 6 Matching

## 6.1 Problem Definitions

An input instance for a matching problem is an undirected graph $G = (N, E)$ of $n$ nodes and $m$ edges. An integer-valued cost $cost(v, w)$ is defined for each edge in $E$.

The matching problem has several formulations, three of which are presented below. Participants in the Implementation Challenge are welcome to choose the version that seems most interesting.

**Minimum-Weight Perfect Matching.** A *perfect matching* $M_p$ is a subset of $E$ in which each node $v$ in the graph is represented exactly once. (It is assumed that $n$ is even). The problem is to find the perfect matching $M_p$ for which

$$\sum_{(v,w)\in M_p} cost(v,w)$$

is *minimized* (or to report that no perfect matching exists).

**Maximum Weighted Matching.** A *matching* $M$ is a subset of $A$ in which each node is represented at most once. (The matching is not necessarily perfect because some nodes may not appear in $M$.)

The problem is to find a matching $M$ such that

$$\sum_{(v,w)\in M} cost(v,w)$$

is *maximized.*

The maximum weighted matching problem is a generalization of the assignment problem to nonbipartite graphs.

**Maximum Matching.** For this problem edge costs are ignored. The problem is to find a matching $M$ such that $|M|$ is maximized. An algorithm for maximum weighted matching can solve a maximum matching problem by setting all edge costs to 1.

## 6.2 Graph Structure.

Graph instances for the matching problems are assumed to have the properties listed below.

- For the minimum-weight perfect matching problem it is assumed that $n$ is even.
- It is assumed that the nodes in the graph have integer names 1 through $n$.
- All costs and coordinates are integer-valued.
- For DIMACS benchmark inputs it is assumed that $cost(v,w)$ is in the range $[-2^{31} \ldots 2^{31}-1]$. Other test inputs may have larger magnitudes.

- No *a priori* limit is set on $n$ or $m$.
- No self-edges appear; that is, for each edge $(v, w)$ it must be the case that $v \neq w$.
- Multiple edges $(v, w)$ between any pair of vertices are not allowed.

## 7 File Formats for Flow Problems

This section describes a standard file format for network inputs and outputs. There is no requirement that participants follow these specifications; however, compatible implementations will be able to make full use of DIMACS support tools. (Some tools assume that output is appended to input in a single file.)

Participants are welcome to develop translation programs to convert instances to and from more convenient, or more compact, representations; the Unix **awk** facility is recommended as especially suitable for this task.

Since the maximum flow and assignment problems are restrictions of minimum-cost flows, instances for the former two can be solved by an algorithm for the latter. Translation programs (written in **awk**) are available from DIMACS which convert input instances for the restricted problems into ones for the general problem.

All files contain ASCII characters. Input and output files contain several types of *lines*, described below. A line is terminated with an end-of-line character. Fields in each line are separated by at least one blank space. Each line begins with a one-character designator to identify the line type.

### 7.1 Input Files

Input files for the three network problems have suffixes **.min**, **.max**, or **.asn** to identify the intended problem. Files are assumed to be well-formed and internally consistent: node identifier values are valid, nodes are defined uniquely, exactly $m$ arcs are defined, and so forth.

- **Comments.** Comment lines give human-readable information about the file and are ignored by programs. Comment lines can appear anywhere in the file. Each comment line begins with a lower-case character **c**.

```
c This is an example of a comment line.
```

- **Problem line.** There is one problem line per input file. The problem line must appear before any node or arc descriptor lines. For network instances, the problem line has the following format.

  ```
  p PROBLEM NODES ARCS
  ```

  The lower-case character `p` signifies that this is the problem line. The `PROBLEM` field contains one of the following three-character problem designators: `min`, `max`, or `asn`. The value in the problem field defines the format for arc and node descriptors. The `NODES` field contains an integer value specifying $n$, the number of nodes in the network. The `ARCS` field contains an integer value specifying $m$, the number of arcs in the network.

- **Node Descriptors.** All node descriptor lines must appear before all arc descriptor lines. The required format for node descriptor lines depends upon the intended problem type:

  - For **minimum-cost flow** instances, the node descriptor lines describe supply and demand nodes, but not transshipment nodes; that is, only nodes with nonzero node flow values $b(v)$ appear. There is one node descriptor line for each such node, with the following format.

    ```
    n ID FLOW
    ```

  - For **maximum flow** instances, exactly two node descriptor lines appear, for the source and sink nodes. They may appear in either order, each with the following format.

    ```
    n ID WHICH
    ```

  - For **assignment** instances, node descriptor lines list the "source" nodes in set $N_1$ only. The format for node descriptor lines is as follows:

    ```
    n ID
    ```

  The lower-case character `n` signifies that this is a node descriptor line. The `ID` field gives a node identification number, an integer between 1 and $n$. The `FLOW` field gives the node flow value $b(v)$. The `WHICH` field gives either a lower-case `s` or `t`, designating the source and sink, respectively.

- **Arc Descriptors.** There is one arc descriptor line for each arc in the network. The required format for arc descriptor lines depends upon the intended problem type.

  - For a **minimum-cost flow** instance, arc descriptor lines are of the following form.

    ```
    a SRC DST LOW CAP COST
    ```

  - For a **maximum flow** instance, arc descriptor lines are of the following form.

    ```
    a SRC DST CAP
    ```

  - For an **assignment** instance, arc descriptor lines are of the following form.

    ```
    a SRC DST COST
    ```

  The lower-case character **a** signifies that this is an arc descriptor line. For a directed arc $(v, w)$ the `SRC` field gives the identification number for the source vertex $v$, and the `DST` field gives the destination vertex $w$. Identification numbers are integers between 1 and $n$. The `LOW` field contains the value of $low(v, w)$, and the `CAP` field gives the value of capacity $cap(v, w)$. The `COST` field contains $cost(v, w)$.

## 7.2 Output Files

For each network problem the solution is an *integer-valued* flow assignment $f(v, w)$. The output file should list the solution and the flow assignment for all arcs $(v, w)$ in $G$. Three types of lines may appear in output files.

- **Comment Lines.** Comment lines are identical in form to those defined for input files. If there is no feasible solution then the algorithm should report this fact on a comment line (in such a case, neither solution lines nor flow lines will appear in the output).

- **Solution Lines.** The solution line contains the solution value: that is, the quantity that was minimized (for minimum-cost flows) or maximized (for maximum flows and assignment). The solution line has the following format.

  ```
  s  SOLUTION
  ```

The lower-case character `s` signifies that this is a solution line. The `SOLUTION` field contains an integer corresponding to the solution value.

- **Flow Assignments.** There is one flow assignment line for each arc in the network. Flow assignment lines have the following format.

```
f  SRC  DST  FLOW
```

  The lower-case character `f` signifies that this is a flow assignment line. For arc $(v, w)$, the `SRC` and `DST` fields give $v$ and $w$, respectively. The `FLOW` field gives the flow assignment $f(v, w)$.

## 8 File Formats for Matching Problems

Two different input formats are defined for matching problems. Participants may choose one or both formats. Files containing graphs defined in geometric format that have the suffix **.geom**. Files containing graphs in the edge-list format have the suffix **.edge**. All files contain ASCII characters. Input and output files contain several types of *lines*, described below. A line is terminated with an end-of-line character. Fields in each line are separated by at least one blank space. Each line begins with a one-character designator to identify the line type.

### 8.1 Geometric Graphs

In a geometric graph, each vertex $v$ is represented by coordinates $(x_1, x_2, \ldots x_d)$ representing a point in some d-dimensional region. The cost of an edge $(v, w)$ is the distance between $v$ and $w$ according to some distance metric. Graphs in the geometric format are assumed to be complete.

- **Comment Lines.** Comment lines give human-readable information about the file and are ignored by programs. Comment lines can appear anywhere in the file. Each comment line begins with a lower-case character `c`.

```
c This is an example of a comment line.
```

  Comments may be used, for example, to suggest a particular distance metric or to give the bounds on coordinate ranges in the data.

- **Problem Line.** The problem line appears before any vertex descriptor lines. The format of the problem line is as follows.

  ```
  p geom NODES DIMENSION
  ```

  The lower-case character `p` signifies that this is a problem line. The lower-case character string `geom` identifies this as a geometric instance for the matching problem. The `NODES` field contains the number of nodes, and the `DIMENSION` field the number of coordinate dimensions *d*.

- **Node Descriptors.** There is one node descriptor line for each node in the graph, each with the following format.

  ```
  v  X1  X2  X3  . .. XD
  ```

  The lower-case character `v` signifies that this is a vertex descriptor line. The fields `X1, X2 .   .   .XD` give the *d* coordinate values for the vertex.

## 8.2 Graphs Represented By Edge Adjacencies

Nongeometric graphs and graphs which are not necessarily complete may be represented by listing edges and edge costs as follows.

- **Comment Lines.** Comment lines give human-readable information about the file and are ignored by programs. Comment lines can appear anywhere in the file. Each comment line begins with a lower-case character `c`.

  ```
  c This is an example of a comment line.
  ```

- **Problem Line.** The problem line appears before any edge descriptor lines. The format of the problem line is as follows.

  ```
  p edge NODES EDGES
  ```

  The lower-case character `p` signifies that this is a problem line. The lower-case character string `edge` identifies this as a instance for the matching problem which lists edge adjacencies. The `NODES` field contains the number of nodes, and the `EDGES` field the number of edges, in the graph.

- **Edge Descriptors.** There is one edge descriptor line for each edge the graph, each with the following format. Each edge $(v, w)$ appears exactly once in the input file and is not repeated as $(w, v)$.

  ```
  e W  V  COST
  ```

  The lower-case character `e` signifies that this is an edge descriptor line. For an edge $(w, v)$ the fields `W` and `V` specify its endpoints. The `COST` field gives the edge cost, $cost(v, w)$.

## 8.3 Output Files

The output of a matching algorithm is a solution value and a list of the edges in the matching. Output files for matching should have the following format.

- **Solution Lines.** The solution line contains the solution value: that is, the quantity that was minimized (for minimum perfect matchings) or maximized (for maximum matching problems). The solution line has the following format.

  ```
  s  SOLUTION
  ```

  The lower-case character `s` signifies that this is a solution line. The `SOLUTION` field contains an integer corresponding to the solution value.

- **Matching Lines.** There is one matching line for each edge in the matching. Each has the following format.

  ```
  m  V W
  ```

  The lower-case character `m` signifies that this is a matching line. For edge $(v, w)$, the `V` and `W` fields give endpoints $v$ and $w$.