A.D. 1308
unipg
DIPARTIMENTO
DI INGEGNERIA

Progetto di

Models and Algorithms for Data Visualization

Corso di Laurea in Ingegneria Informatica e Robotica

Curriculum Data Science – A.A. 2024-2025

Dipartimento di Ingegneria

docente

Prof. Giuseppe Liotta

MarvelNet

Explore the connections between heroes and their movie appearances

363433 **Gian Marco Ferri** gianmarco.ferri@studenti.unipg.it

Contents

1 Introduction

The MarvelNet project is an interactive data visualization system designed to explore the connections between heroes of the Marvel Universe and their appearances in movies within the Marvel Universe through filtering, zooming, dragging and node details.
The main purpose of this visualization system is to provide an engaging and insightful platform for users to understand and analyze the complex web of relationships within the Marvel Universe.

The data that will be visualized includes a list of Marvel characters who appear in various movies, a list of movies that feature these heroes, and relationships between heroes and the movies they appear in, represented as links between nodes.
The dataset is structured as an XML file (`marvel.xml`) that contains nodes for heroes and movies, and edges that define the connections between them. Each node has attributes such as id, type (hero or movie), and name, while edges have source and target attributes that indicate the connections.

The main reasons to visualize these data are to provide Marvel fans, researchers, and data enthusiasts with an intuitive tool to explore and understand the connections within the Marvel Universe and allow them to discover interesting patterns and relationships among heroes and movies, such as which heroes frequently appear together and the centrality of certain characters in the Marvel narrative.
For these exact reasons, this project offers an interactive platform where users can dynamically explore the data, adjust visualization parameters, and filter the information to suit their interests and inquiries.

The following sections are organized as follows:

- **Requirements Gathering**: This section describes the dataset in detail, the end-users of the visualization system, the data modeling processes, and the task modeling approach. It outlines how the data are extracted, cleaned and transformed, as well as the needs and tasks of the end-users.
- **Design**: This section covers the design phase of the project, including visualization design, interaction design, architectural and technological choices, and algorithm engineering. Describes the choices made for visual encoding, interaction operations, system architecture, and the algorithms used to implement visualization.
- **Realization**: This section provides a detailed description of the system development and user interface. Describes the implementation process, the

components of the user interface, and offers visual examples of the system in action.

2 Requirements Gathering

Description of the Dataset

The dataset used in the MarvelNet project is an XML file named `marvel.xml`, taken from the 2019 Graph Drawing Contest and available here. It encapsulates information about characters (heroes) and movies within the Marvel Universe, along with the relationships between them. The dataset is structured into nodes and edges:

- **Nodes**: Represent individual entities, either heroes or movies. Each node has attributes such as id, type, and name.
- **Edges**: Represent connections between nodes, indicating which heroes appear in which movies. Each edge has source and target attributes that define the relationship between two nodes.

Example Structure of the Dataset

```
<graphml>
  <key attr.name="type" attr.type="string" for="node" id="type" />
  <key attr.name="name" attr.type="string" for="node" id="name" />
  <graph edgedefault="directed" id="">
    <!-- Nodes representing heroes -->
    <node id="h0">
      <data key="type">hero</data>
      <data key="name">Ant-Man</data>
    </node>
    <!-- Nodes representing movies -->
    <node id="m0">
      <data key="type">movie</data>
      <data key="name">Ant-Man</data>
    </node>
    <!-- Edges representing connections -->
    <edge source="h0" target="m0" />
  </graph>
</graphml>
```

This XML structure provides a clear and organized way to represent the relationships within the Marvel Universe, making it suitable for visualization.

2.1 End-users

The end-users of the MarvelNet visualization system include:

- **Marvel Fans**: Enthusiasts who are deeply interested in the Marvel Universe and want to explore the connections between their favorite heroes and movies.
- **Data Enthusiasts**: Individuals who have a keen interest in data visualization and network analysis, and who enjoy exploring complex datasets in a visual format.
- **Researchers**: Academics and professionals who are studying the interconnectedness of characters in the Marvel Universe, possibly for sociocultural studies or media analysis.

The needs of these end-users are diverse: Marvel fans seek to see which heroes appear in which movies, understand the relationships and interactions among different heroes, and explore the Marvel Universe visually and interactively.
Data enthusiasts aim to analyze the network structure of the Marvel Universe, discover interesting patterns and relationships within the dataset, and use the visualization as a case study for network analysis and data visualization techniques.
Researchers are interested in gaining insights into the narrative structure and character dynamics of the Marvel Universe, utilizing the visualization for academic or professional studies on the impact of media franchises, and exploring the data interactively to extract meaningful conclusions for their research.

2.2 Data Modeling

2.2.1 Data Extraction

- **Process**: The XML dataset is loaded using D3.js, a JavaScript library for producing dynamic, interactive data visualizations in web browsers. D3.js provides functions to parse XML data, making it easy to extract the necessary information.
- **Implementation**: The data is extracted using D3's `d3.xml` function, which loads the XML file and parses it into a format that can be manipulated programmatically.

2.2.2 Data Cleaning and Transformation

- **Process**: Once the data is extracted, it undergoes a cleaning and transformation process to ensure it is in a suitable format for visualization.
- **Implementation**:
  - **Parsing Nodes**: Each node in the XML file is parsed to extract attributes such as `id`, `type`, and `name`.
  - **Parsing Edges**: Each edge is parsed to extract the `source` and `target` attributes, defining the connections between nodes.

- **Transformation**: The parsed data is transformed into a JSON format that D3.js can use to create the visualization.

2.2.3 Definition of Dataset Types and Attribute Types

- **Nodes**:
  - **Attributes**:
    - `id`: A unique identifier for each node.
    - `type`: Specifies whether the node is a hero or a movie.
    - `name`: The name of the hero or movie.
- **Edges**:
  - **Attributes**:
    - `source`: The `id` of the source node (hero).
    - `target`: The `id` of the target node (movie).

2.3 Task Modeling

The tasks are formulated to define user interactions with the visualization system. Each task is represented as an {action, target} pair:

Tasks

1. **Explore Connections**:
   - **Action**: Explore
   - **Target**: Connections between heroes and movies
   - **Description**: Users can navigate through the graph to see which heroes appear in which movies, allowing them to explore the network of connections.
2. **Identify Number of Connections for each Hero and Movie**:
   - **Action**: Identify
   - **Target**: Number of connections for each hero and for each movie
   - **Description**: Users can click on a hero or movie node to see how many movies that hero appears in or which heroes appear in that movie, respectively; providing a quick way to identify the prominence of each hero in the Marvel Universe.

3. **Discover Relationships Among Heroes**:

   - **Action**: Discover
   - **Target**: Relationships among heroes
   - **Description**: Users can analyze the network to discover how different heroes are connected through their movie appearances, revealing the underlying relationships and interactions.

3 Design

3.1 Visualization Design

3.1.1 Marks and Channels

The visualization uses the following marks and channels to represent the data:

Marks

- **Nodes**: Representing heroes and movies in the Marvel Universe.
- **Edges (Links)**: Representing the connections between heroes and the movies they appear in.

Channels

- **Color**: Different colors are used to distinguish between heroes and movies. For example:
  - **Heroes**: Represented by steel blue circles.
  - **Movies**: Represented by green circles.
- **Size**: Node size can be dynamic based on the number of connections (degree). This helps to visually emphasize nodes with more connections.
- **Position**: Determined by the force-directed layout algorithm, which positions nodes based on their connections to minimize overlap and make the network structure clear.

3.1.2 Semantics

- **Nodes**: Each node represents either a hero or a movie. The type of node (hero or movie) is indicated by its color.
- **Edges**: Each edge represents a connection between a hero and a movie, indicating that the hero appears in that movie.

3.1.3 Visual Encoding Idioms

A **force-directed graph** was selected as the primary visual encoding idiom. This selection is motivated by several factors:

- **Clarity**: The force-directed layout algorithm arranges the nodes in a way that reduces overlap and makes the connections between nodes clear and easy to follow.
- **Interactivity**: This layout allows for interactive exploration, where users can drag nodes, zoom in and out, and pan across the graph to explore different parts of the network.
- **Aesthetics**: The dynamic nature of the force-directed layout makes the visualization engaging and visually appealing, which enhances the user experience.

3.2 Interaction Design

3.2.1 Interaction Operations

The designed interaction operations include:

- **Zooming and Panning**: Users can zoom in and out of the graph using mouse scroll or touch gestures, and pan across the graph by dragging. This allows users to focus on specific parts of the network or get an overview of the entire graph.
- **Hovering**: When users hover over a node, the connected edges and related nodes are highlighted. This helps users quickly see the connections and relationships of a particular node.
- **Clicking**: Clicking on a node displays detailed information about the hero or movie, such as the name, type, and number of connections. This interaction provides deeper insights into the selected node.
- **Filtering**: Users can use checkboxes to filter the nodes displayed in the graph, such as showing only heroes or only movies. This allows users to customize the visualization based on their interests.
- **Dynamic Sizing**: A toggle allows users to enable or disable dynamic sizing of nodes based on their degree (number of connections). This helps users identify the most connected nodes easily.

3.2.2 Interaction Paradigm

The chosen interaction paradigm is **direct manipulation**, where users interact directly with the visualization elements. This paradigm is effective because it provides immediate feedback and makes the interaction intuitive and engaging. Users can manipulate nodes, zoom, pan, and filter the graph in real-time, enhancing their exploration and understanding of the data.

3.3 Architectural and Technological Choices

3.3.1 System Type

The system being designed is a **web-based interactive visualization system**. It is accessible via a web browser and provides a dynamic and interactive user experience.

3.3.2 Technologies

The technologies used in this project include:

- **HTML**: For structuring the web page.
- **CSS**: For styling the web page and visualization elements.
- **JavaScript**: For implementing the interactive functionality.
- **D3.js**: For creating and manipulating SVG elements and implementing the force-directed graph layout. D3.js is chosen for its powerful data visualization capabilities and flexibility.

3.3.3 Techniques and Tools

The techniques and tools provided to the user for interacting with the system include:

- **Force-directed Graph Layout**: To visualize the network of connections between heroes and movies.
- **D3.js**: To handle data binding, transitions, and interactions.
- **HTML and CSS**: To create the user interface and style the visualization.
- **Interactive Controls**: Sliders and checkboxes for adjusting visualization parameters and filtering the data.

3.4 Algorithm Engineering

This subsection describes the algorithms chosen or designed to implement the visualization and interaction idioms for the MarvelNet project.

3.4.1 Force-directed Layout Algorithm

The force-directed layout algorithm positions nodes dynamically based on their connections. It uses forces such as repulsion and attraction to arrange the nodes in a way that minimizes overlap and makes the network structure clear.

Implementation The force-directed layout algorithm is implemented using D3.js's `forceSimulation`. The simulation includes:

- `forceLink`: Defines the links between nodes with a specified distance.
- `forceManyBody`: Applies a repulsive force between nodes to prevent overlap.
- `forceCenter`: Centers the graph in the SVG container.

```
const simulation = d3.forceSimulation(graphData.nodes)
    .force('link', d3.forceLink(graphData.links).id(d => d.id).distance(100))
    .force('charge', d3.forceManyBody().strength(-200))
    .force('center', d3.forceCenter(width / 2, height / 2));
```

3.4.2 Drag Behavior

The drag behavior allows users to interactively manipulate node positions. It is implemented using D3.js's `drag` behavior.

Implementation

```
node.call(d3.drag()
    .on('start', dragstart)
    .on('drag', dragging)
    .on('end', dragend));

function dragstart(event, d) {
    if (!event.active) simulation.alphaTarget(0.3).restart();
    d.fx = d.x;
    d.fy = d.y;
}

function dragging(event, d) {
    d.fx = event.x;
    d.fy = event.y;
}

function dragend(event, d) {
    if (!event.active) simulation.alphaTarget(0);
    d.fx = null;
    d.fy = null;
}
```

3.4.3 Zoom and Pan Algorithms

Zoom and pan algorithms allow users to navigate the network by zooming in and out and panning across the graph. These interactions are implemented using D3.js's `zoom` behavior.

Implementation

```
const zoom = d3.zoom().on('zoom', function(event) {
    svg.attr('transform', event.transform);
});

d3.select('#graph').call(zoom);
```

By integrating these algorithms, the MarvelNet visualization system provides a dynamic and interactive experience, allowing users to explore the Marvel Universe in an engaging and insightful way.

4 Realization

4.1 System Development

The development of the MarvelNet visualization system involved several key stages, each crucial for ensuring a robust, interactive, and user-friendly experience. Below is a detailed account of the development process:

Initial Setup

1. **Environment Setup**:
    - Installed necessary tools and libraries, including D3.js for data visualization.
    - Set up a local development environment using a code editor such as Visual Studio Code.
2. **Loading and Parsing Data**:
    - Used D3.js to load the XML dataset (`marvel.xml`).
    - Implemented a function to parse the XML data, extracting nodes (heroes and movies) and edges (connections).

```
d3.xml('Dataset/marvel.xml').then(function(xml) {
    const graphData = parseGraphData(xml);
    console.log("Parsed Graph Data:", graphData);

    // Calculate and display statistics
    displayStatistics(graphData);

    // Create the graph visualization
    createGraph(graphData);
}).catch(function(error) {
    console.error('Error loading or parsing XML data:', error);
});
```

Data Transformation

- Transformed the parsed data into a JSON format compatible with D3.js for visualization purposes.

Graph Creation

- Created an SVG element as the container for the graph.
- Implemented the force-directed layout using D3.js to position nodes dynamically.

```
const simulation = d3.forceSimulation(graphData.nodes)
    .force('link', d3.forceLink(graphData.links).id(d => d.id).distance(100))
    .force('charge', d3.forceManyBody().strength(-200))
    .force('center', d3.forceCenter(width / 2, height / 2));
```

Node and Link Representation

- Represented nodes as circles, with different colors for heroes and movies.
- Represented edges as lines connecting the nodes.

```
const node = svg.selectAll('.node')
    .data(graphData.nodes)
    .enter().append('circle')
    .attr('class', d => `node ${d.type}`)
    .attr('r', 10)
    .call(d3.drag()
        .on('start', dragstart)
        .on('drag', dragging)
        .on('end', dragend))
    .on('mouseover', mouseover)
    .on('mouseout', mouseout)
    .on('click', function(event, d) {
        displayNodeDetails(d);
        event.stopPropagation();
    });

const link = svg.selectAll('.link')
    .data(graphData.links)
    .enter().append('line')
    .attr('class', 'link')
    .style('stroke', '#999')
    .style('stroke-width', 2)
    .style('opacity', 0.6);
```

Interactive Features

- Implemented zooming and panning to allow users to navigate the graph.
- Added tooltip functionality to display node details on hover.
- Used sliders and checkboxes for filtering and adjusting the visualization.

```
const zoom = d3.zoom().on('zoom', function(event) {
    svg.attr('transform', event.transform);
});
d3.select('#graph').call(zoom);
```

Final Touches

- Applied CSS to style the nodes, links, and tooltips for a polished look.
- Ensured the layout was responsive to different screen sizes.
- Performed extensive testing to ensure all interactive features worked as expected.
- Debugged issues related to data parsing, graph layout, and user interactions.
- Deployed the visualization system to a web server, making it accessible online.
- Created a dedicated project page with links to the system and demonstration videos.

4.2 User Interface

The user interface of the MarvelNet visualization system is designed to be intuitive and user-friendly. Below are the key components of the UI:

Graph Visualization

- **SVG Container**: The main area where the force-directed graph is rendered. It dynamically adjusts to the screen size and allows for zooming and panning.
- **Nodes**: Represent heroes and movies, color-coded for easy identification. Heroes are depicted in steelblue, while movies are in green.
- **Links**: Lines connecting nodes, representing the appearances of heroes in movies.

Interactive Controls

- **Zoom and Pan**: Users can zoom in and out using the mouse wheel or touch gestures and pan across the graph by dragging.
- **Node Details**: Hovering over a node highlights its connections and displays a tooltip with details such as the name, type, and number of connections.
- **Filters**: Checkboxes to show or hide heroes and movies, allowing users to focus on specific parts of the network.
- **Dynamic Sizing**: A toggle to enable or disable dynamic node sizing based on the number of connections.

Information Display

- **Statistics Panel**: Displays overall statistics such as the total number of connections, heroes, and movies.
- **Node Information Panel**: Shows detailed information about the selected node, including its name, type, and connections.

4.3 Visual Aids and Live System

The live system and the demonstration video are accessible through the following links:

- MarvelNet Live System
- Demonstration Video

This project can be found at the following link: Github link.

Here are some snapshots of the system in action:

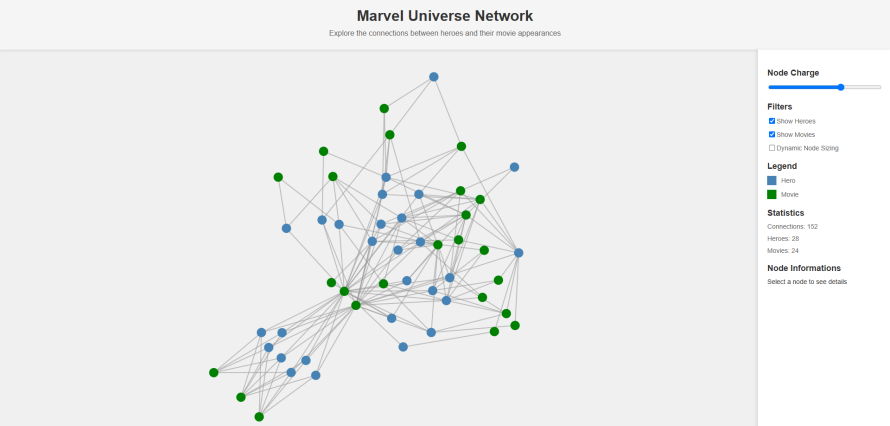

Figure 1: Overview of the MarvelNet user interface.

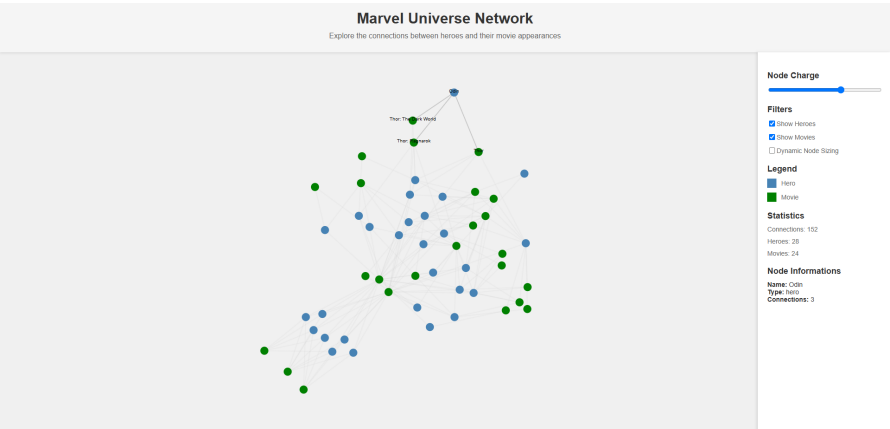

Figure 2: Detailed information displayed upon hovering over a node and clicking it.

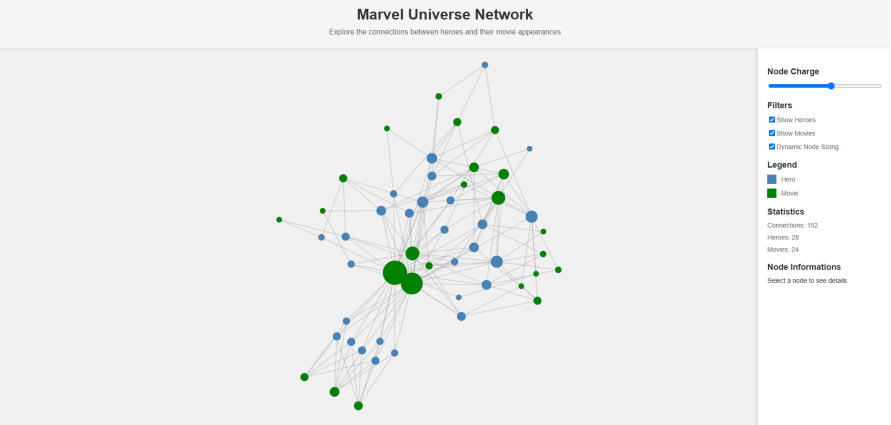

Figure 3: Dynamic sizing of nodes based on their degree.

These resources provide a comprehensive view of how the MarvelNet visualization system works and its interactive features.

By following these detailed steps and utilizing the provided resources, users can fully explore and understand the Marvel Universe through the MarvelNet visualization system.