

INDIVIDUAL REPORT FOR THE COURSE COMP1110 – GROUP D08

ZHAN HO JACOB SHING

1. INDIVIDUAL CONTRIBUTIONS

Throughout this project, I was responsible for the following aspects:

- **Problem Modelling:**

I came up with the rough idea of modelling the campus with the common approach of a graph. Afterwards, I finalised more details on the graph representation (e.g., undirected/directed graph, simple graph/multigraph, attributes that edges and nodes should have, etc.) and implemented the fundamental data classes in Python.

- **Core Implementation:**

I was responsible for implementing the core functionality of the Navigator app, including the design of UI/UX and the code of the pathfinding engine. Apart from coding, “implementation” also includes deciding on the technical stack, project architecture, development workflow, etc.

- **Report Writing:**

For the final report, I was responsible for writing the sections on problem definition and implementation details.

- **Typesetting:**

While other parts of the report were written by other members, they were written on a shared Google Docs file. To maintain a consistent style and format across the report, and provide a more professional presentation, I was responsible for typesetting the entire report in L^AT_EX.

2. PERSONAL EVALUATION

Overall, I consider the project to be a success in terms of its computational and algorithmic design. The campus navigation problem involves a number of interacting concerns, including heterogeneous traversal types, user-configurable preferences, speed-dependent costs, and multi-waypoint journey planning. The graph-theoretic formulation adopted in this project reduces these concerns to a single coherent model, in which all of the above are expressed as properties of nodes, edges, and a parameterised cost function. This transformation enables the application of Dijkstra’s algorithm in a clean and well-understood way, and the resulting system produces correct and meaningful routes across a range of scenarios.

That said, I believe the project has significant room for improvement in terms of practical applicability. The data model captures only the most fundamental attributes of each campus segment: its type and a single base traversal time. In reality, the traversability and cost of a segment depend on a much richer set of factors. For instance, the model currently does not account for time-of-day variation in crowd density, the directional asymmetry of escalators, the waiting time associated with lifts, or the temporary unavailability of infrastructure due to maintenance. A user relying on the system during peak hours or when a particular lift is out of service would receive estimates that are noticeably inaccurate.

Addressing these limitations would not necessarily require a fundamental redesign of the algorithmic core, since the cost function is already parameterised and could be extended to accept additional inputs. The primary challenge lies in the data collection and maintenance effort required to populate the additional attributes reliably. Nonetheless, bridging this gap between the theoretical model and the practical conditions of a real campus would be the most important direction for improving the system.

3. REFLECTION

3.1. What I Learnt. One of the most valuable lessons from this project was the experience of transforming a loosely stated real-world problem into a formal, computable model. Deciding on the appropriate data structure, identifying which attributes are essential, and expressing the cost function in precise mathematical terms required a level of rigour that I had not previously applied to a self-directed project.

This process gave me a much clearer understanding of how abstract mathematical concepts such as weighted graphs and shortest-path algorithms connect to practical engineering decisions.

On the technical side, I gained familiarity with the Textual library for building terminal user interfaces in Python. Prior to this project, I had not worked with any terminal UI framework, and learning to structure a multi-screen application with reactive state management was a new experience. Although my use of the library remained at a basic level, it gave me a concrete understanding of the component model and how UI state can be decoupled from business logic.

Working on a group software project also consolidated my understanding of collaborative development practices. In particular, I set up a GitHub Actions workflow to run linting checks automatically on each push, which helped enforce a consistent code style across contributions from different members. This was my first experience configuring a continuous integration pipeline for a shared Python repository, and it reinforced the importance of automated tooling for maintaining codebase consistency as a project grows.

In terms of AI usage, this project provided an opportunity to reflect more carefully on the ethics of using AI-assisted tools in an academic context. Specifically, I developed a clearer personal framework around the two responsibilities I consider most important: acknowledging AI involvement transparently, and independently verifying that the generated outputs are correct and appropriate before accepting them. These principles, which are discussed in more detail in the following section, are ones I intend to carry forward in future work.

Finally, as a side endeavour, typesetting the project plan, final report, and other documents in L^AT_EX gave me the opportunity to considerably consolidate my L^AT_EX skills. Handling multi-page tables, custom column types, floating figures, and landscape layouts within a document of this complexity was a meaningful exercise in its own right.

3.2. Reflections on the Project Process. On the whole, I found this project to be an enjoyable and rewarding experience. There is a particular satisfaction in taking an idea that exists only as an informal description and seeing it realised as a working application that produces sensible and useful results. The fact that the system we built is directly relevant to our own daily campus experience made the goal feel concrete and worthwhile throughout the development process.

I also found it notable that the course explicitly permitted the use of AI tools, and I think this decision contributed meaningfully to the quality of the project experience. The benefit was not simply that certain tasks became less time-consuming, though that was certainly true. More significantly, working with state-of-the-art AI tools in a structured project context provided firsthand experience of how such tools fit into a real development workflow, including both their capabilities and their limitations. It also created a genuine opportunity to engage with the ethical questions surrounding AI use in an academic setting, which I believe is a more effective form of learning than encountering those questions in the abstract.

4. DECLARATION ON USAGE OF ARTIFICIAL INTELLIGENCE

It is hereby declared that tools based on artificial intelligence and large language models were used in the development of the project and the writing of reports. The tools were used for various purposes, including but not limited to code generation, debugging, simplify the boilerplate, etc. Details of the tools used, prompts given, and outputs received are attached in [section A](#). Specific use cases and how were the results generated by AI tools are verified are discussed in this section.

4.1. Using AI for code generation. AI tools were used to generate code for implementing the project (details in [subsection A.1](#)). To ensure the quality and correctness of the generated code, instead of giving a one-sentence prompt and accepting the output, I carefully considered the detailed outcomes that I expect for the project, and listed them in an instruction file. This allows the generated code to have a better quality on first pass.

After the code was generated, I executed the application and manually tested all of the features with normal and edge cases, while noting down the faulty outcomes. As presented in the follow-up prompts, I then fed the error messages back to the AI tool for correction. This iterative process of testing and debugging was carried out until the application was stable and up to the expected standard.

Apart from my verification, the code was also tested or reviewed by other members of the group, which further ensures the correctness and quality of the code.

4.2. Using AI for boilerplate generation. For some parts of the project, such as writing the final report, there was repetitive overhead that was not directly related to the core learning outcomes of the project. For example, for illustrating the scenario study outcomes, figures need to be drawn for multiple solutions, and they share the same style and format. Manually writing the TikZ code for each figure is

time-consuming and not necessarily related to the purpose of this project. To address this, I used AI tools to generate templates for the TikZ figures, which can then be easily reused for multiple figures with minor modifications.

As shown in [subsection A.2](#), AI generated a clearly documented and easy-to-use template for drawing solution diagrams, which significantly reduced the time and effort needed to create the figures for the report. The generated template was compiled and verified to work correctly. All four figures in the final report were created based on the template or its variants.

4.3. Using AI for report writing. For writing the problem definition part of the report, the group has already developed a rough idea about using graphs to model the campus navigation problem (as can be seen in the project plan). However, for presenting the definition and modelling in a more formal and professional way, I used AI tools to read through the original ideas and formulate them mathematically with proper notations. Since the generated content was based on the original ideas and consists of elementary mathematical formulae, it was verified by reading through the generated content and ensuring the mathematics remain consistent with the idea and the codes.

APPENDIX A. PROMPTS USED WITH AI

This section lists the prompts used to generate content with the assistance of AI tools, along with details of the tools used and the outputs (where appropriate).

A.1. Prompts for generating the main_app module.

- **Tool:** GitHub Copilot (Visual Studio Code extension)
- **Model:** Claude Opus 4.5
- **Mode of Operation:** Agent mode
- **Thinking Effort:** High

The first prompt is:

- **Prompt:**
"Read and follow the instructions."
- **Outputs:** (Thinking process omitted.)
"Implementation complete. Here's a summary of what was built:
 - route_engine.py: Business logic: Dijkstra pathfinding with speed multiplier (per edge-type exponents), preference-based avoidance, and multi-waypoint routing. Generates candidate routes using 4 strategies (user prefs, no stairs, prefer lifts, fewest stops) with deduplication and auto-labeling.
 - main_app.py: MainApp class holding shared state (campus map, waypoints, preferences, speed multiplier).
 - screens/configuration_screen.py { Three-panel layout: waypoint list with add/remove/move, 4 preference checkboxes (with accessible→avoid stairs + prefer lifts auto-toggle), speed multiplier ± 0.1 controls, and validated Go button.
 - screens/waypoint_selection_screen.py { Alphabetically sorted list with search filter, excludes already-selected waypoints.
 - screens/solutions_screen.py { Scrollable route cards with time/stops summary and colored badges.
 - screens/route_details_screen.py { Step-by-step waypoint list with edge type and time info.
 4 incremental commits made following the feat: convention."

Files changed/created: all files under campus_nav/main_app/

- **Attachments:**
- INSTRUCTIONS.md

```

1 # LLM Instructions
2
3 This document provides instructions for an LLM agent to build an application.
4 The agent should follow these instructions to finish the implementation.
5
6 ## General Guidelines
7
8 1. Read the entire instructions carefully before starting the implementation.
9 2. Read the entire codebase to understand the existing structure and how the new
   feature should fit in.
10 3. Follow the coding style and conventions (including commit message format)
   used in this repository.
11 4. Write clean, modular code. Documentation should be only at an appropriate
   level and should be concise. No verbose comments.
12 5. As you build, make incremental commits. Before committing, ensure that the
   code is working but no unit tests are required. Do not make big commits that
   include multiple features or refactors. Each commit should be focused on a
   single change or addition.
13 6. When implementing, if you encounter any ambiguities or uncertainties in the
   instructions, ask for clarification before proceeding. Do not make
   assumptions that are not stated in the instructions.
14

```

```
15 ## Task Overview
16
17 You will be writing a Python app with the module name 'campus_nav.main_app'.
18 The app's class name should be 'MainApp'.
19 This app will be similar to Google Maps but for a campus.
20 Users can select start and end points on the campus map, and toggle on/off
    different preferences (e.g. avoid stairs, etc.), and optionally add
    intermediate points.
21 The app will then calculate several candidate routes and display them as a list.
22 The app should be implemented using Textual and completely text-based (no
    graphical map rendering is required).
23 The app should have several screens.
24 The details of the implementation and features are described below.
25
26 ### Architecture
27
28 1. Create a self-contained module named 'campus_nav.main_app' and expose
    necessary classes and functions in its '__init__.py'.
29 2. The app will depend on Textual. Data for the campus map and data classes are
    already defined in the 'campus_nav.models' module. Use those, and extend
    them INTERNALLY as needed, but do not modify the existing data classes,
    unless they have bugs.
30 3. The app should be structured with a main 'MainApp' class that manages the
    overall application flow and state, and separate screen classes for each
    distinct screen in the app (e.g. 'RouteSelectionScreen', 'RouteDisplayScreen
    ', etc.).
31 4. The app should not be built with GUI frameworks. It should be entirely text-
    based using Textual's widgets and layout system. Custom widgets can be
    created.
32 5. Arrange files in submodules as needed for better organization, but keep the
    overall structure simple and intuitive. For example, you might have a '
    screens' submodule for the different screen classes, and a 'widgets'
    submodule for any custom widgets you create. '.tcss' files should live
    within the same directory as the '.py' files that use them, with the same
    base name (e.g. 'main_app.py' and 'main_app.tcss').
33 6. Business logic need to be separated from UI code as much as possible. The
    screen classes should primarily handle rendering and user interaction, while
    the route calculation logic should be in separate functions or classes.
34
35 ### Screens
36
37 Name the screens as appropriate, but they must all be suffixed with 'Screen'.
38
39 1. **Configuration Screen**: This is the first screen that the app launches into
    . It will have three sections, arranged horizontally:
40     - Waypoint selection: A list of waypoints (start, intermediate, end) that
        the user can edit. The user can have at most 5 points (including start
        and end). Points must be distinct. There should be buttons at the bottom
        to add/remove points, and buttons to move points up/down the list. Each
        point in the list can be focused to highlight and modified using the
        buttons. No edit button is needed, just select the point and use the add
        /remove/move buttons.
41     - Preferences: A list of toggleable preferences (e.g. avoid stairs, prefer
        lifts, accessible route, etc.).
42     The app should handle preferences that are mutually exclusive automatically
        if any.
```

```

43 - Configuration: A customisable "speed multiplier" set at 1.0 by default,
    that the user can adjust to simulate faster or slower walking speeds. A
    larger multiplier means faster walking, so the time cost for each edge
    will be adjusted accordingly in the route calculations. Also, different
    edge types should receive different intensity of the multiplier
    adjustment (e.g. stairs might be more affected by the speed multiplier
    than flat paths and lifts are not affected at all).
44 - A "Go" button at the bottom that becomes enabled when the configuration is
    valid (e.g. at least start and end points are set, preferences are
    consistent, etc.). The button spans the entire width of the screen and
    is centered horizontally.
45
46 2. Waypoint Selection Screen: This screen presents a list of all available
    waypoints, sorted alphabetically, with a search input at the top to filter
    the list. The list items are selectable, and can be confirmed with a "
    Confirm" button at the bottom. When a waypoint is selected and confirmed,
    the app should return to the Configuration Screen and update the selected
    waypoint. Only those waypoints that are not already selected in the
    Configuration Screen should be selectable in this screen.
47
48 3. Solutions Screen: This screen displays the calculated routes based on the
    user's configuration.
49 On top-left there should be a "Back" button to return to the Configuration
    Screen.
50 The main area of the screen should be a scrollable list of route options, each
    showing a summary of the route (e.g. total time, number of stops, etc.) and
    some labels or badges indicating the features of the route (e.g. has stairs,
    no stairs, fastest, least stops, etc.).
51 There is a "Details" button for each route that the user can select to view more
    detailed information about the route.
52
53 4. Route Details Screen: This screen shows the details of a selected route
    from the Solutions Screen. It should include a top-down list of waypoints in
    the route, with edge information between them (e.g. time cost, edge type,
    etc.). There should be a "Back" button to return to the Solutions Screen.

```

Then, with a follow-up prompt:

- **Prompt:** (Exception message omitted.)

"When I click on the add button, I get:

...

BadIdentifier: 'wps.Library Extension' is an invalid id; identifiers must contain only letters, numbers, underscores, or hyphens, and must not begin with a number."

- **Outputs:** (Thinking process omitted.)

"The node ID Library Extension has a space, which is invalid in Textual widget IDs. I'll stop using node IDs as widget IDs and use list indices instead.

The fix was committed successfully. The issue was that node IDs containing spaces (like Library Extension) were being used directly as Textual widget IDs, which only allow letters, numbers, underscores, and hyphens.

The fix tracks filtered nodes in a separate `_filtered_nodes` list and uses the `ListView` index to look up the corresponding node ID, avoiding widget IDs derived from node names entirely."

Files changed: `campus_nav/main_app/screens/waypoint_selection_screen.py`

Then, with a follow-up prompt:

- **Prompt:**

"For the waypoints list, disable the click to edit. It is currently conflicting with click to select. Only allow editing via the buttons.

Add a new preference option "prioritise stairs" in the config screen"

- **Outputs:** (Thinking process omitted.)

"Done. Removed the on_list_view_selected handler so clicking/pressing Enter on the waypoints list no longer opens the picker. Added an "Edit" button that uses the currently highlighted list item to open waypoint selection.

Added "Prioritise Stairs" as a new preference. It halves the cost of stair edges during pathfinding so routes with stairs are favoured. It's mutually exclusive with "Avoid Stairs" and gets auto-disabled when "Accessible Route" is checked."

Files changed: campus_nav/main_app/route_engine.py and campus_nav/main_app/screens/configuration_screen.py

Finally:

- **Prompt:**

"For the accessible path option, escalators should be avoided. And, for screens that have a back button, add a keyboard binding for the button. For the app selector, the confirm button is not centered correctly, fix it."

- **Outputs:** (Thinking process omitted.)

"Now I have all the context. Three changes needed:

Accessible path: also avoid escalators - set avoid_escalators checkbox when accessible is toggled
Back button bindings: add escape binding on SolutionsScreen and RouteDetailsScreen
App selector button: the margin: 0 50 0 50 is too large and not truly centered - use align instead
Three changes applied:

Accessible route now also checks "Avoid Escalators" and disables it (alongside avoid stairs / prefer lifts)
Escape key bound to go back on both SolutionsScreen and RouteDetailsScreen
App selector button centering fixed - switched from margin: 0 50 to width: 50% with align-horizontal: center"

Files changed: campus_nav/main_app/screens/configuration_screen.py, campus_nav/main_app/screens/solutions_screen.py, campus_nav/main_app/screens/route_details_screen.py

A.2. Prompts for generating TikZ template.

- **Tool:** GitHub Copilot (Visual Studio Code extension)
- **Model:** Claude Sonnet 4.6
- **Mode of Operation:** Agent mode
- **Thinking Effort:** High

The prompt is:

- **Prompt:**

"Help me setup an easy to use template in this file that I can copy to other figures. I need to draw diagrams to represent the solutions that my project has produced. You should define styles for nodes (I need at least a dot and the node name in monospaced fonts). There might be parallel paths between nodes so the nodes need to be a wide rounded corner rect. with two ends being semi circles (basically like a railway map). Also define styles for directed edges. Each edge can also take extra text information next to it. Since I will be presenting multiple solutions generated by my project in one single figure for the same set of inputs, you need to define multiple sets of edge styles (diff colours will be enough) for me to easily use."

- **Outputs:** The following template file was generated:

```
1 \documentclass{standalone}
2 \usepackage{tikz}
```

```

3 \usetikzlibrary{shapes.misc, arrows.meta, positioning, bending}
4
5 %% Helper: pill-shaped node label -- a filled dot followed by a monospaced name.
6 %% Usage inside a [station] node: \stlabel{NODE_ID}
7 \newcommand{\stlabel}[1]{\textbullet\enspace\texttt{#1}}
8
9 \begin{document}
10 \begin{tikzpicture}[
11     %% -- Node styles -----
12     %%
13     %% station: pill/stadium shape (semi-circular ends, like a metro map).
14     %% Use \stlabel{NAME} as the node content to get the dot + monospaced text.
15     %% Example:
16     %% \node[station] (A) {\stlabel{MTR_A2}};
17     station/.style={
18         draw,
19         shape=rounded rectangle, % from shapes.misc -- true semi-circular ends
20         rounded rectangle arc length=180,
21         minimum height=1.8em,
22         inner xsep=0.9em,
23         align=center,
24         thick,
25         fill=white,
26     },
27     %% -- Base directed edge style -----
28     %%
29     %% All per-solution styles inherit from this.
30     %% For parallel edges use bend left/right:
31     %% \draw[sol1, bend left=20] (A) to node[elabel, above] {label} (B);
32     route/.style={
33         ->,
34         >=Stealth[round],
35         semithick,
36         shorten >=4pt,
37         shorten <=4pt,
38     },
39     %% -- Per-solution edge styles (one colour per solution) -----
40     %%
41     %% Usage: \draw[sol1] (A) -- (B);
42     %% \draw[sol2, bend left=20] (A) to node[elabel, above] {text} (B);
43     sol1/.style={route, color=blue!80!black}, % Solution 1 -- blue
44     sol2/.style={route, color=red!70!black}, % Solution 2 -- red
45     sol3/.style={route, color=green!55!black}, % Solution 3 -- green
46     sol4/.style={route, color=orange!85!black}, % Solution 4 -- orange
47     sol5/.style={route, color=violet!80!black}, % Solution 5 -- violet
48     %% -- Edge label style -----
49     %%
50     %% Place inline on an edge:
51     %% \draw[sol1] (A) -- node[elabel, above] {5 min / Walk} (B);
52     elabel/.style={font=\small\itshape, inner sep=2pt, fill=white},
53 ]
54
55 %% -- Place nodes here -----
56
57 \node[station] (UStr_Upper_MTR_Exit) {\stlabel{UStr\_Upper\_MTR\_Exit}};
58 \node[station, below=10pt of UStr_Upper_MTR_Exit] (UStr_Upper_InnoWing_Entrance) {\stlabel{UStr\_Upper\_InnoWing\_Entrance}};

```

```

59 \node[station, below=3cm of UStr_Upper_InnoWing_Entrance] (StephenH_GeoMuseum_
    Entrance) {\stlabel{StephenH\GeoMuseum\Entrance}};
60
61 %% -- Draw edges here -----
62
63 %% Solution 1 (blue):
64 \draw[sol1] (UStr_Upper_MTR_Exit) -- node[elabel, above] {Walk 3 min} (UStr_Upper_
    InnoWing_Entrance);
65 \draw[sol1] (UStr_Upper_InnoWing_Entrance) -- node[elabel, above] {Walk 2 min} (
    StephenH_GeoMuseum_Entrance);
66
67 % Solution 2 (red) -- offset with bend to avoid overlapping sol1:
68 \draw[sol2, bend left=25] (UStr_Upper_MTR_Exit) to node[elabel, above] {Bus 5 min} (
    StephenH_GeoMuseum_Entrance);
69
70 \end{tikzpicture}
71 \end{document}

```

Then, with a follow-up prompt:

- **Prompt:**

"For the nodes, I want to put a small rounded rectangle as the "dot", then the name on the right of the dot, and align the nodes by the rounded rectangle. The name should not be inside the rectangle. Make the dot smaller, line thicker, and wider.

Add a box for legends. Then I can place legend items inside the box relatively and I can declare where the box is and everything is in the right place. In the legend box, each item should first be a coloured arrow pointing right, then a label solution N, and below the label I can write other texts I want"

- **Outputs:** The template file was updated to the following:

```

1 \documentclass{standalone}
2 \usepackage{tikz}
3 \usetikzlibrary{shapes.misc, arrows.meta, positioning, bending, fit}
4
5 \begin{document}
6 \begin{tikzpicture}[
7     %% -- Node styles -----
8     %%
9     %% station: small pill/stadium shape used as the "dot" indicator.
10    %%   The node body is EMPTY; the monospaced name lives OUTSIDE as a label.
11    %%   Edges and positioning are anchored to the pill shape, not the text.
12    %%
13    %%   Usage:
14    %%     \node[station, label={\stlabel}right:NODE\_ID] (A) {};
15    %%     \node[station-bold, label={\stlabel-bold}right:NODE\_ID] (A) {};
16    %%
17    station/.style={
18        draw,
19        shape=rounded rectangle, % from shapes.misc - true semi-circular ends
20        rounded rectangle arc length=180,
21        minimum height=0.9em, % height sets the semicircle diameter
22        minimum width=2.4em, % must exceed height to get a flat middle section
23        inner xsep=0pt,
24        inner ysep=0pt,
25        line width=1.6pt,
26        fill=white,
27    },
28    %% station-bold + stlabel-bold: use on start/end/waypoint nodes for emphasis.

```

```

29 station-bold/.style={station, line width=3pt},
30 %% stlabel: monospaced name label placed to the right of the station dot.
31 %% Always pass it as the style argument to the label option:
32 %% label={stlabel}right:NAME}
33 stlabel/.style={font=\ttfamily, inner sep=2pt},
34 stlabel-bold/.style={stlabel, font=\ttfamily\bfseries},
35 %% -- Base directed edge style -----
36 %%
37 %% All per-solution styles inherit from this.
38 %% For parallel edges use bend left/right:
39 %% \draw[sol1, bend left=20] (A) to node[elabel, above] {label} (B);
40 route/.style={
41   ->,
42   >=Stealth[round],
43   semithick,
44   % shorten >=4pt,
45   % shorten <=4pt,
46 },
47 %% -- Per-solution edge styles (one colour per solution) -----
48 %%
49 %% Usage: \draw[sol1] (A) -- (B);
50 %% \draw[sol2, bend left=20] (A) to node[elabel, above] {text} (B);
51 sol1/.style={route, color=blue!80!black}, % Solution 1 - blue
52 sol2/.style={route, color=red!70!black}, % Solution 2 - red
53 sol3/.style={route, color=green!55!black}, % Solution 3 - green
54 sol4/.style={route, color=orange!85!black}, % Solution 4 - orange
55 sol5/.style={route, color=violet!80!black}, % Solution 5 - violet
56 %% -- Edge label styles -----
57 %%
58 %% Use the per-solution variant (elabel1-elabel5) so that identical labels
59 %% on parallel edges are still distinguishable by colour.
60 %%
61 %% For N parallel edges on the same node pair, spread them symmetrically:
62 %% N=2: bend left=15 / bend right=15
63 %% N=3: bend left=25 / straight (no bend) / bend right=25
64 %% N=4: bend left=35 / bend left=12 / bend right=12 / bend right=35
65 %%
66 %% To additionally separate labels, use xshift on the node option:
67 %% node[elabel3, right, xshift=8pt]
68 %%
69 %% Example (2 parallel edges):
70 %% \draw[sol1, bend left=15] (A) to node[elabel1, left] {label} (B);
71 %% \draw[sol2, bend right=15] (A) to node[elabel2, right] {label} (B);
72 %% -- Legend styles -----
73 %%
74 %% legendtitle: bold coloured solution name - pair with the solution colour.
75 %% e.g. \node[legendtitle, text=blue!80!black] {Solution 1};
76 legendtitle/.style={font=\small\bfseries, inner sep=0pt, anchor=west},
77 %% legenddesc: smaller grey text for the lines below the title.
78 legenddesc/.style={font=\small, text=black!60, inner sep=0pt, anchor=west},
79 elabel/.style={font=\small\ttfamily, inner sep=5pt},
80 elabel1/.style={elabel, text=blue!80!black},
81 elabel2/.style={elabel, text=red!70!black},
82 elabel3/.style={elabel, text=green!55!black},
83 elabel4/.style={elabel, text=orange!85!black},
84 elabel5/.style={elabel, text=violet!80!black},
85 ]
86

```

```

87 %% -- Place nodes here -----
88
89 % omitted
90
91 %% -- Draw edges here -----
92
93 %% Solution 1 (blue):
94 % omitted
95
96 %% Solution 2 (red) - parallel to sol1 on shared segments, bent symmetrically:
97 % omitted
98
99 %% Solution 3 (green) - parallel to sol1 on shared segments, bent symmetrically:
100 % omitted
101
102 %% -- Legend -----
103 %%
104 %% Move the whole legend by changing the shift value.
105 %% Each item occupies a vertical slot; the slot height is 2.8em.
106 %% Copy/remove item blocks as needed and update the fit= list on the box node.
107 %%
108 %% Item anatomy (all x-coordinates are relative to the scope origin):
109 %% 0 - arrow start
110 %% 1.8em - arrow end / label start offset
111 %% 2.2em - text column (title + description nodes)
112 %% slot y - 0, -2.8em, -5.6em, -8.4em, -11.2em (for items 1-5)
113 %%
114 \begin{scope}[shift={(-6.5cm, -1cm)}] %% change this to reposition the legend
115
116 %% -- Item 1 --
117 \node[inner sep=0, minimum size=0] (leg-orig) at (0, 0) {}; %% left-edge anchor
118 \draw[sol1] (0, 0) -- (1.8em, 0);
119 \node[legendtitle, text=blue!80!black] (li1-title) at (2.2em, 0.4em) {Solution
120 1};
121 \node[legnddesc] (li1-desc) at (2.2em, -0.4em) {Fastest,
122 3m 44s};
123
124 %% -- Item 2 --
125 \draw[sol2] (0, -2.8em) -- (1.8em, -2.8em);
126 \node[legendtitle, text=red!70!black] (li2-title) at (2.2em, -2.8em+0.4em) {
127 Solution 2};
128 \node[legnddesc] (li2-desc) at (2.2em, -2.8em-0.4em) {4m
129 38s, no stairs};
130
131 %% -- Item 3 --
132 \draw[sol3] (0, -5.6em) -- (1.8em, -5.6em);
133 \node[legendtitle, text=green!55!black] (li3-title) at (2.2em, -5.6em+0.4em) {
134 Solution 3};
135 \node[legnddesc] (li3-desc) at (2.2em, -5.6em-0.4em) {4m
136 41s, no stairs};
137
138 %% -- Bounding box - update fit= when adding/removing items --
139 \node[draw, rounded corners=4pt, inner sep=8pt,
140 fit=(leg-orig)(li1-title)(li1-desc)(li2-title)(li2-desc)(li3-title)(li3-
141 desc)
142 ] {};
143 \end{scope}

```

```
138 |
139 | \end{tikzpicture}
140 | \end{document}
```

A.3. Prompts for generating the problem definition part of the report.

- **Tool:** GitHub Copilot (Visual Studio Code extension)
- **Model:** Claude Sonnet 4.6
- **Mode of Operation:** Agent mode
- **Thinking Effort:** High

The prompt is:

- **Prompt:**
"Write two subsections under this section. One is problem definition, take a look at the attached project guidelines and what we have already proposed in the project plan, and also the source code of the project, expand our problem definition and fit it into the report. Another is problem modeling, take a look at the source code, talk about how the problem is modeled mathematically. Use some math notations to make it more formal and rigorous. You should take a look at the python code and extract the relevant information."
- **Outputs:** *Contents under Problem Definition and Core Implementation in the group report. Note that some contents have been modified manually after generation.*
- **Attachments:** a plaintext file derived from COMP1110ProjectGuidelines.pdf and the source code of COMP1110-D08-Project-Report.pdf