Final Documentation - Wells Assignment Project

Piotr Krzemiński 305785
Zofia Wrona 305803

1 Modifications

The initial documentation has been revised and updated based on the requests specified in the emails until the submission date. As a result, the source code in the initial documentation closely mirrors the provided source code up until the delivery of the final documentation. The only key modification involves rounding float distances to two decimal places and multiplying them by 100 for conversion to integers. This adjustment is implemented to mitigate floating-point comparison errors.

2 User's manual

2.1 Requirements

The application requires the Python executable version 3.7 with numpy version $>=$ 1.21 and matplotlib version $>=$ 3.0.0.

2.2 Execution Guidelines

To execute the application with the provided argument parser, follow the steps outlined below:

Command-Line Options

The application supports the following command-line options:

- `-m, -mode`: Specifies the application mode. The default is `GENERATE_AND_RUN`. Options include:
  - `GENERATE_INPUT`: Generates a new input file.
  - `GENERATE_AND_RUN`: Generates a new input file and runs the Hungarian algorithm.
  - `READ_INPUT`: Reads the provided input file and runs the Hungarian algorithm.
  - `BENCHMARK`: Executes the Hungarian algorithm with each pair of parameters $(wells\_count, houses\_per\_well)$, where $wells\_count \in [1, n]$ and $houses\_per\_well \in [1, k]$.
- `-n`: Sets the value for the number of wells. The default value is 3, and you can adjust it by providing a positive integer value. Note that a combination of (n, k) with values greater or equal to 5 can result in the runtime of around an hour for complex cases.
- `-k`: Sets the value for the number of houses per well. The default value is 3, and you can adjust it by providing a positive integer value. Note that a combination of (n, k) with values greater or equal to 5 can result in a runtime of around an hour for complex cases.
- `-i, -input_file`: Specifies the input file name for the application. The default file name is `input.txt`.
- `-o, -output_file`: Specifies the output file name for the application. The default file name is `output.txt`.

Example Usage

To run the application with custom settings, use the following command:

```
python main.py -m <MODE> -n <n> -k <k> -i <file.txt> -o <file.txt>
```

As an example, this is how the main function is executed for default parameters:

```
python main.py -m GENERATE_AND_RUN -n 5 -k 5 -i input.txt -o output.txt
```

Adjust the values and file paths according to your specific requirements. Any missing argument will take its predefined default value.

3 Tests

To validate the algorithm's correctness, we conducted multiple tests as described below. Running the command:

```
python tests.py
```

executes these tests and saves the outputs to the ***Pictures*** folder.

1. Example from the initial documentation. Runtime of around a second.

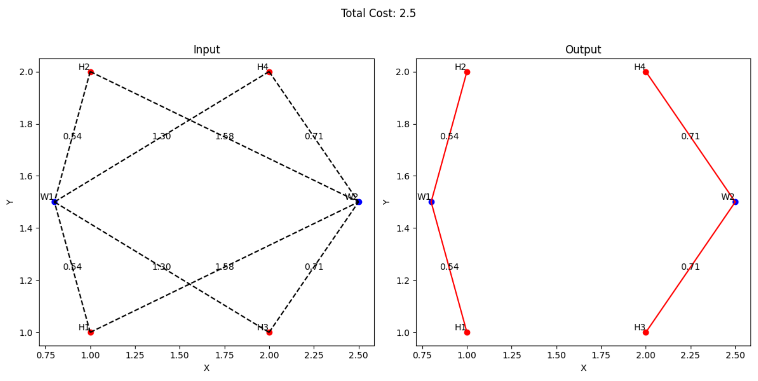

Figure 1: Initial documentation example

2. Randomly generated example with 3 wells and 2 houses per well - verification of correctness for multiple wells. Runtime of around 5 seconds.

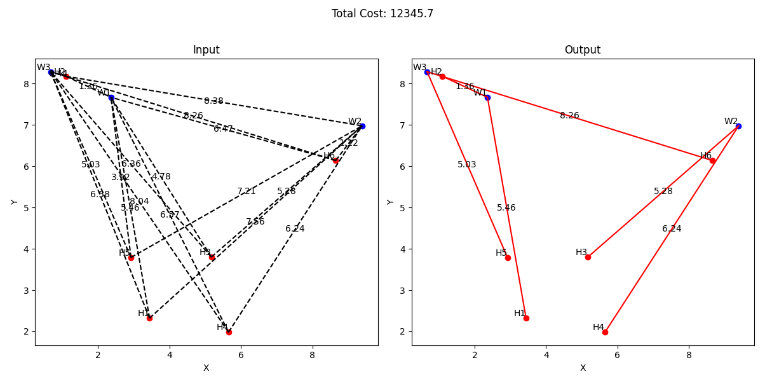

Figure 2: 3 wells, 2 houses per well

3. Randomly generated example with 2 wells and 4 houses per well - verification of correctness for multiple houses per well. Runtime of around 5 seconds.

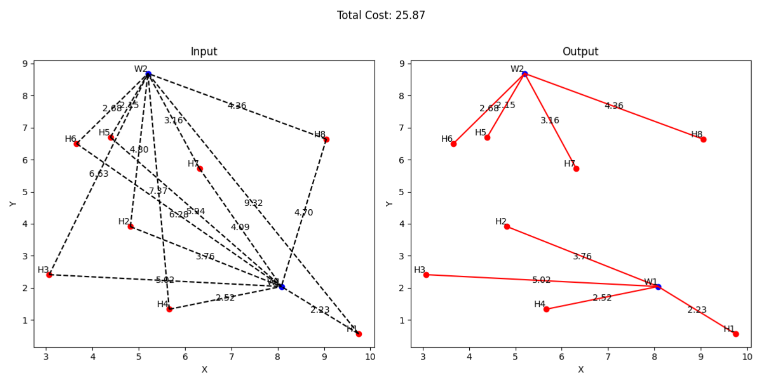

Figure 3: 2 wells, 4 houses per well

4. Large example with 4 wells and 3 houses per well. Runtime of around 30 seconds.

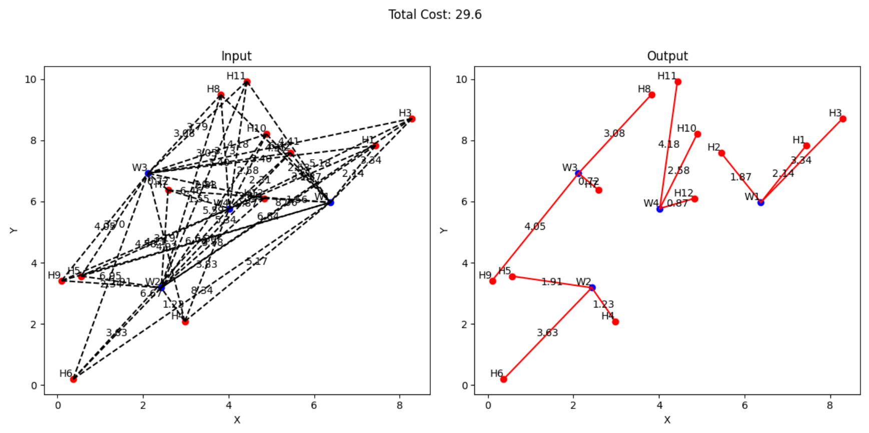

Figure 4: 4 wells, 3 houses per well

The final example is with 5 wells and 5 houses per well. This is a special, simple example for which the runtime was astonishingly low - 40 seconds. The average runtime of the normal 5x5 execution is on average around one hour.

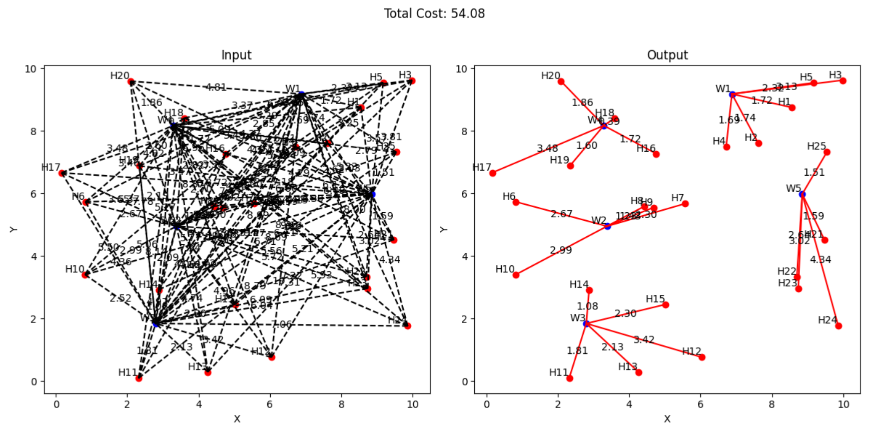

Figure 5: 5 wells, 5 houses per well

The above examples correctly assigned houses to wells, confirming the algorithm's accuracy.

4 Time Complexity Measurements

We measured the execution time for each $(wells\_count, houses\_per\_well)$ pair, with $wells\_count \in [1, 5]$ and $houses\_per\_well \in [1, 5]$. The resulting surface, depicted in green, is shown in Figure 6. We compared it with the predicted worst-time complexity $O((nk)^4)$, represented by the red surface.

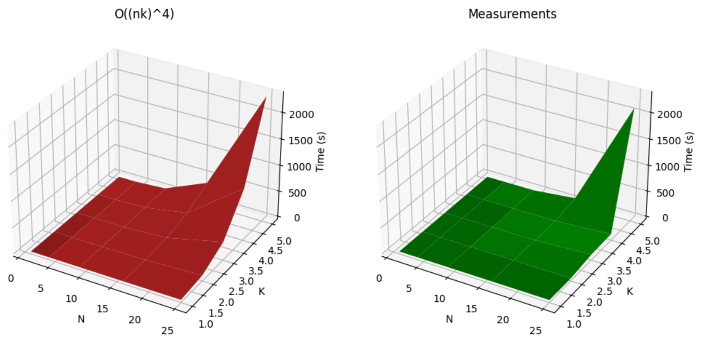

Figure 6: Standard time complexity measurements

Additionally, Figure 7 displays the logarithmic time complexity measurements for better clarity.

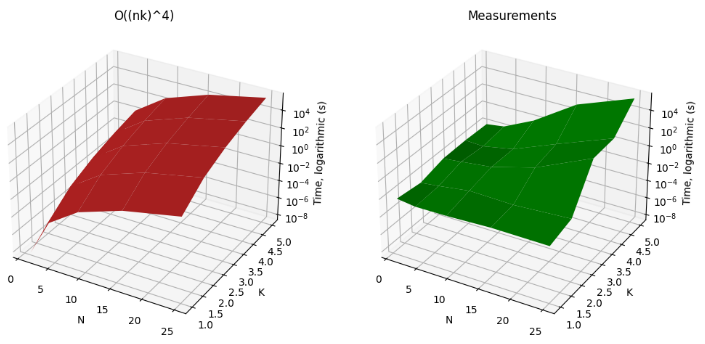

Figure 7: Logarithmic time complexity measurements

The plots affirm that the worst-case runtime of the algorithm aligns with the expected $O((nk)^4)$ complexity.

5 Work Division

We strategically divided the workload to maintain an equal distribution of 50%, and both authors agree that this objective was successfully achieved.

Name	**Responsibilities**
Zofia	Implement the Hungarian algorithm
	Prepare initial documentation: • Correctness and optimality proofs • Mathematical formulation • Matrix pseudo-code
	Improve code clarity and optimize execution
	Conduct tests for correctness verification
	Prepare final documentation: • Test results • Division of work
Piotr	Implement the Hungarian algorithm
	Prepare initial documentation: • Time complexity proof • Problem formulation • Graph pseudo-code
	Create visualizations for input and output
	Create visualizations for time complexity
	Prepare final documentation: • Execution instructions • Time complexity

Table 1: Work division

6 Conclusions

In conclusion, the modified Hungarian algorithm implemented in this project has demonstrated its reliability and efficiency in solving the Wells Assignment problem. The adjustment, involving rounding float distances and converting to integers, enhances the algorithm's stability. Rigorous testing across various scenarios affirms its correctness and scalability, with a time complexity of $O((nk)^4)$. Future improvements could focus on optimizing time complexity, as various sources noted that by utilizing label caching the runtime could be reduced to $O((nk)^3)$.