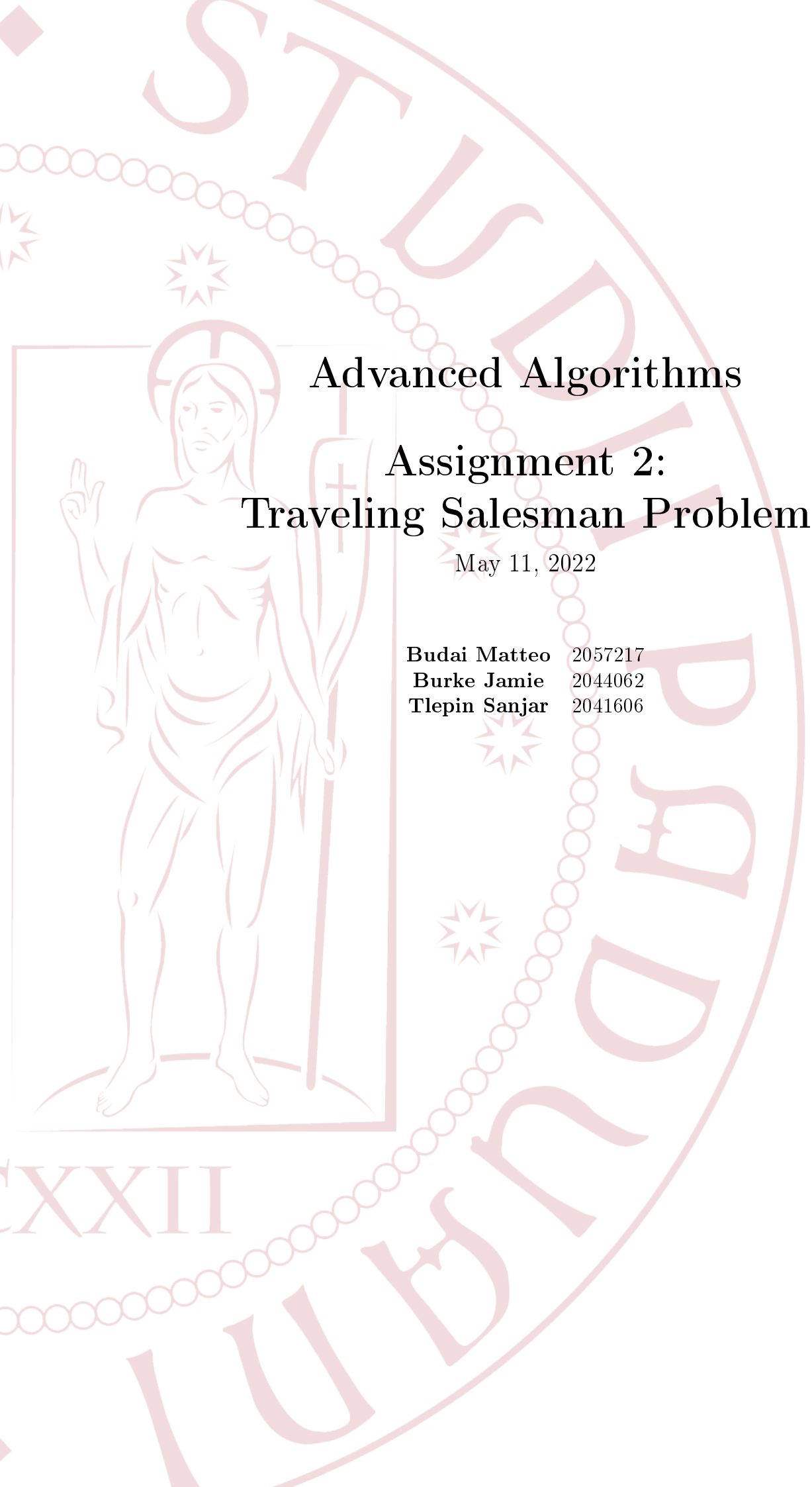

Advanced Algorithms

Assignment 2: Traveling Salesman Problem

May 11, 2022

Budai Matteo	2057217
Burke Jamie	2044062
Tlepin Sanjar	2041606

Contents

1 Introduction **2**

2 Nearest Neighbor Algortihm **3**
2.1 Functions 3
2.2 Implementation 3

3 Random Insertion Algortihm **5**
3.1 Input and Data Structures 5
3.2 Implementation 5
3.3 Complexity 5

4 2-approximate Algorithm **6**
4.1 Input and Data Structures 6
4.2 Implementation 6
4.3 Complexity 6

5 Results **7**
5.1 Table of results 7
5.2 Graph of the execution time comparison 7
5.3 Graph of the error comparison 7

6 Conclusion **8**

1 Introduction

For this assignment, we implemented and analyzed the running times and the errors of three algorithms. The algorithms implemented are:

1. Nearest Neighbor Algorithm
2. Random Insertion Algorithm
3. 2-approximate Algorithm

2 Nearest Neighbor Algortihm

As first constructive heuristics we have chosen Nearest Neighbor. Its goal is to add the neighbor with the minimum weight not in the path. The steps are the following:

1. **Initialization:** start from the single-node path 0;
2. **Selection:** let (v0, ..., vk) be the current path. Find the vertex vk+1 not in the path with minimimun distance from vk;
3. **Insertion:** insert vk+1 immedaitely after vk;
4. repeat from (2) until all vertices are inserted in the path.

2.1 Functions

The implementation doesn't require particular data structure so we created two functions for the algorithm:

- **NearestNeighbor(g,v)**: method that builds the hamiltonian circuit and returns its weight and the time;
- **minDist(p,g,l)**: method that finds the neighbor with the minimum weight between the last vertex insert in the path p and its neighbors that are in some indexes of the list g.

2.2 Implementation

The final solution is the following:

- We start the time;
- We execute a deep copy unvisited of the original list v;
- We follow the algorithm and we start from the node 0 *(startingNode)* of the unvisited list.
- We remove the starting node from the list unvisited and we add always the starting node in our path *(is a list)*;
- We initialize our solution *(dist)* to zero and we start our cycle until we have visited all vertex *(len(unvisited)>0))*;
- In the while cycle:
    1. We find the neighbor with the minimum distance with the function minDist:
        - It initalize the distance to infinite, the vertex to none and it take the last node of the path *(p)*;
        - In the cycle it checks all the neighbors of the node and returns the distance and the vertex with the minimum distance that is not just in p.

2. We remove the vertex found from the list unvisited and we add the vertex in the path;
3. We add the distance of the vertex found and we continue with the next iteration until we removed all the vertex from unvisited.

- When we finish the while cycle we add to dist the distance between the last node in the path and the startingNode and to close the hamiltonian cycle we add the vertex to the path;
- We end the time and we calculate the time cost;
- We return the solution and the time.

3 Random Insertion Algortihm

3.1 Input and Data Structures

3.2 Implementation

3.3 Complexity

4 2-approximate Algorithm

4.1 Input and Data Structures

4.2 Implementation

4.3 Complexity

5 Results

5.1 Table of results

5.2 Graph of the execution time comparison

5.3 Graph of the error comparison

6 Conclusion

how do the algorithms behave with respect to the various instances? Is there an algorithm that always manages to do better than the others with respect to the approximation error? Which of the three algorithms you have implemented is more efficient?