CMPE 300: Analysis of Algorithms

Project 1 - Hamiltonian* Path

Yasin Başt uğ *
Erencem Özbey †

Deadline: 15 November 2025, 12:00 (Noon)

1 Introduction

This project is designed to assess your understanding of the core concepts in algorithm analysis. Besides you will be expected to critically think on a well-known algorithm and optimize it for the given conditions. You will be given pseudocodes of the input generation algorithm and a naive version of the Hamiltonian* path-finding algorithm. In the first part of this project, your task is to analyze this path finding algorithm thoroughly, determine its basic operation(s), calculate its best, worst, and average-case time complexities, execute it, and compare the theoretical results with the actual execution times. In the second part, you need to find a better algorithm considering the input characteristics and analyze its time complexity. There is an additional bonus section to further improve these algorithms and discuss the bounds on the time complexity for this problem.

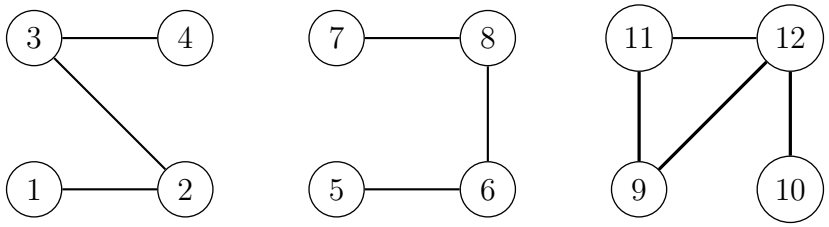
1.1 Problem Definition (Hamiltonian* Path)

We say there exists a **Hamiltonian* path** between vertices *start* and *end* in an undirected graph G with $3n$ nodes if there is a path that starts at *start*, ends at *end*, and it visits exactly n distinct nodes (all nodes should be different).

We provide a graph generator that builds a compound graph consisting of three separate subgraphs with size n. Since the Hamiltonian* path has length n, *start* and *end* nodes must lie in the same subgraph. If *start* and *end* are in different components, a Hamiltonian* path *cannot* exist.

*osman.bastug@std.bogazici.edu.tr
†erencem.ozbey@std.bogazici.edu.tr

Example:



- Hamiltonian* path $11 \to 10$ exists
- Hamiltonian* path $6 \to 7$ doesn't exist
- Hamiltonian* path $1 \to 8$ doesn't exist

2 Algorithms

2.1 Graph Construction Algorithm (Three Subgraphs)

The generator creates three random subgraphs, each of size n, with *no cross edges between the subgraphs*. A random permutation is then applied to the vertex indices to hide the underlying subgraph structure, and the algorithm outputs the (*start*, *end*) node indices.

You must use the provided script `graph_construction.py` to generate the graph and obtain the corresponding start–end node indices.

Algorithm 1 GenerateTrickyGraph(n)

$N \leftarrow 3n$
start $\leftarrow$ RandomInteger(0, $N-1$)
repeat
    end $\leftarrow$ RandomInteger(0, $N-1$)
until *end* $\neq$ *start*
Initialize *graph* as an $N \times N$ zero matrix ▷ Adjacency matrix
$A \leftarrow \{0, 1, \ldots, n-1\}$ ▷ component 1 (size n)
$B \leftarrow \{n, n+1, \ldots, 2n-1\}$ ▷ component 2 (size n)
$C \leftarrow \{2n, 2n+1, \ldots, 3n-1\}$ ▷ component 3 (size n)
AddConnectedSubgraph(*graph*, A)
AddConnectedSubgraph(*graph*, B)
AddConnectedSubgraph(*graph*, C)
perm $\leftarrow$ RandomPermutation($0, 1, \ldots, N-1$) ▷ Hide the structure
Initialize *permuted* as an $N \times N$ zero matrix
for $i = 0$ **to** $N-1$ **do**
    for $j = 0$ **to** $N-1$ **do**
        permuted[i][j] $\leftarrow$ *graph*[*perm*[i]][*perm*[j]]
    end for
end for
graph $\leftarrow$ *permuted*
start $\leftarrow$ IndexOf(*start*, *perm*)
end $\leftarrow$ IndexOf(*end*, *perm*)
return (*graph*, *start*, *end*)

Algorithm 2 AddConnectedSubgraph($graph$, V)

$L \leftarrow$ list of vertices in V
$L \leftarrow$ RandomPermutation(L) ▷ Spanning path to guarantee connectivity
for $k = 0$ **to** $|L| - 2$ **do**
    $u \leftarrow L[k]$; $v \leftarrow L[k+1]$
    $graph[u][v] \leftarrow 1$; $graph[v][u] \leftarrow 1$
end for
for each unordered pair $\{u, v\}$ with $u, v \in V$ and $u < v$ **do**
    if $graph[u][v] = 0$ **and** Bernoulli($\frac{1}{2}$) **then**
        $graph[u][v] \leftarrow 1$; $graph[v][u] \leftarrow 1$
    end if
end for

2.2 Naive Hamiltonian* Path Algorithm (to Analyze First)

You will first analyze a naive approach that tries all length **n** permutations of the **3n** vertices, restricted to those beginning at *start* and ending at *end*, checking adjacency between consecutive vertices.

Algorithm 3 Hamiltonian*Naive ($3n \times 3n$ Graph, $start, end$)

$V \leftarrow \{0, 1, \ldots, 3n-1\}$; $N \leftarrow 3n$
for each subset $S \subseteq V$ **with** $|S| = n$ **and** $\{start, end\} \subseteq S$ **do**
    Let L be the list of vertices in S in any fixed order
    Build $H \in \{0,1\}^{n \times n}$ where $H[a][b] \leftarrow graph[L[a]][L[b]]$ for all a, b
    $s \leftarrow$ index in L where $L[s] = start$
    $t \leftarrow$ index in L where $L[t] = end$
    if AllPermutations(H, s, t) **then**
        return True
    end if
end for
return False

Algorithm 4 AllPermutations(H, s, t)

$n \leftarrow$ number of rows/cols of H
for each permutation $perm$ of $\{0, 1, \ldots, n-1\}$ **with** $perm(0) = s$ and $perm(n–1) = t$ **do**
    if Hamiltonian*Check(H, $perm$) **then**
        return True
    end if
end for
return False

Algorithm 5 HAMILTONIAN*CHECK(H, $perm$)

```
n ← number of rows/cols of H
for i = 0 to n − 2 do
    if H[perm(i)][perm(i + 1)] = 0 then
        return False                          ▷ early exit on first missing edge
    end if
end for
return True                                   ▷ all consecutive pairs are adjacent
```

3 Analysis for the Naive Algorithm (50 pt)

3.1 Theoretical Analysis (25 pt)

- What is the basic operation of the naive algorithm? (5 pt)
- What is the time complexity of the naive algorithm? Give the rigorous mathematical analysis for the best case, worst case, and average case. In your analysis, do not consider the graph construction. (20 pt)

3.2 Experimental Analysis (25 pt)

- Implement the code in Python, try at least 5 different n values where $n \geq 4$. Create an "n vs execution time" plot. Analyze the plot and compare your results with your theoretical analysis. (25 pt)

Note 1: For the execution time, do not consider graph constructions. For each n value, take the average of 10 rounds.

Note 2: You need to submit your code files as well to be evaluated for this part.

4 Analysis of the Optimized Algorithm (50 pt)

You now need to exploit the graph's structure:

As mentioned in the graph construction algorithm, the main graph has three separate subgraphs. Use this property to enhance the naive algorithm. You will still need to call Algorithm 4 as a subroutine.

4.1 Theoretical Analysis (25 pt)

- What is the time complexity of this new algorithm? Is it different from the Naive algorithm? Give the rigorous analysis for the best case, worst case, and average case. In your analysis, do not consider the graph construction.

4.2 Experimental Analysis (25 pt)

- Implement your algorithm in Python, try 10 different n values where $n \geq 4$, and create an "n vs execution time" plot. Analyze the plot and compare your results with your theoretical analysis and the Naive algorithm.

Note 1: For each n value, take the average of 10 rounds. Do not consider graph constructions for the execution time.

Note 2: You need to submit your code files as well to be evaluated for this part.

5 Bonus (20 pt)

For the bonus part you need to implement a superior algorithm (in time complexity). Unlike the last section it is not required to call Algorithm 4 in this part.

- Implement the algorithm and perform an average case analysis. (10 pt)
- Explain your algorithm and experimentally show that this is superior to the previous algorithms. (5 pt)
- Discuss the bounds on the time complexity for this problem. Can we find a polynomial-time algorithm? (5 pt)

Note: You need to submit your code to be evaluated for the first two bullet points.

6 Submission Format

Submit a single ZIP archive named: `StudentNo1_StudentNo2.zip`.
The ZIP must have the following layout:

```
StudentNo1_StudentNo2.zip
-- StudentNo1_StudentNo2/
    |-- Report.pdf
    |-- solution.py
```

Report:

- File name: `Report.pdf`.
- Present your theoretical analysis and experimental results.

Code:

- File name: `solution.py`.
- Provide the following functions:

```
hamiltonian_naive(graph, start, end)
hamiltonian_optimized(graph, start, end)
hamiltonian_bonus(graph, start, end)
```

- Each function should return True/False, indicating whether a Hamiltonian* path exists from `start` to `end`.

Notice 1: The deadline is strict. No late submissions will be accepted.

Notice 2: Use of Large Language Models (LLMs) is strictly prohibited.