CSC4120 Spring 2024 - Written Homework 11

Yohandi 120040025
Andrew Nathanael 120040007

May 8, 2024

Problem 1.

Reduce the 3-coloring problem to the k-coloring problem for any $k > 3$. For example, show how to reduce the 3-coloring problem to the 8-coloring problem.

Given a graph $G = (V, E)$ for which we need to determine if it can be colored with 3 colors, we construct a new graph G' as follows:

- Start with G as part of G', i.e., $V(G')$ initially contains all vertices of V, and $E(G')$ contains all edges of E.
- Append a complete graph K_{k-3} to G'. This complete graph is composed of $k-3$ vertices, where each pair of vertices is connected by an edge. In 8-coloring problem, the complete graph is K_5.
- Connect each vertex in V to every vertex in K_{k-3}, to ensure that no vertex in V can share a color with any vertex in K_{k-3} since all vertices of K_{k-3} are adjacent to each vertex in V and to each other.

Claim that G' can be colored with 8 colors if and only if G can be colored with 3 colors.

($\Rightarrow$) If G' can be colored with 8 colors, consider the restriction of this coloring to the vertices of G. Since all vertices of K_{k-3} (which is part of G') must be distinctly colored from each other due to their pairwise connectivity and each must also be distinctly colored from any vertex in V (as every vertex in V is connected to every vertex in K_{k-3}), we are left with $k-(k-3) = 3$ colors available for the vertices in G. Hence, if G' can be 8-colored, G must be 3-colorable.

($\Leftarrow$) Conversely, if G can be colored with 3 colors, then these colors can be used to color the vertices of G within G'. The remaining 5 colors (since $k = 8$ and $k-3 = 5$) can be used to color the K_5 graph. Because there are no edges between distinct vertices of K_{k-3} and G beyond those connecting each to all, this coloring scheme suffices for G'.

Both the construction and argument show how a solution to the 8-coloring problem on G' provides a solution to the 3-coloring problem on G. This reduction is valid for any $k > 3$ by adjusting the size of K_{k-3}. Thus, solving the k-coloring problem for G' also solves the 3-coloring problem for G.

Problem 2.

Recall the traveling salesman problem:

```
TSP
Input: A matrix of distances; a budget b
Output: A tour which passes through all the cities and has length <= b,
        if such a tour exists.
```

The optimization version of this problem asks directly for the shortest tour.

```
TSP-OPT
Input: A matrix of distances
Output: The shortest tour which passes through all the cities.
```

Show that if TSP can be solved in polynomial time, then so can TSP-OPT.

Given a problem TSP that decides if there exists a tour of all cities with total distance less than or equal to a given budget b and a problem TSP-OPT that requires finding the minimum distance tour that visits all cities. We are to show if TSP is polynomial-time solvable, then TSP-OPT is also polynomial-time solvable.

Suppose:

- $d_{\max}$ is the maximum value in the distance matrix (maximum distance between any two cities).
- n is the number of cities. Then, the upper bound for any tour in the worst-case scenario is $n \cdot d_{\max}$, when the tour uses the maximum distance repeatedly.
- $d_{\min}$ is the minimum value in the distance matrix (excluding zeros, which represent no direct path between cities or the same city). Then, the lower bound for any tour is $n \cdot d_{\min}$.

Problem reduction:

- Perform binary search on tour length by considering range $[n \cdot d_{\min}, n \cdot d_{\max}]$. The decision of the binary search is as follows:
  - If a tour of length `mid` exists, then record it as a potential answer and search the range $[\texttt{left}, \texttt{mid} - 1]$.
  - If a tour of length `mid` doesn't exist, search the range $[\texttt{mid} + 1, \texttt{right}]$.
- The result of the binary search will be the smallest possible value of b such that a tour of length $\leq b$ exists. Since each iteration of the binary search halves the search space, the number of iterations required would be $\mathcal{O}(\log(n \cdot d_{\max} - n \cdot d_{\min}))$.

If each decision that calls `TSP`, for which the `TSP` can be solved in polynomial time, then the problem reduction `TSP-OPT` can also be solved in polynomial time as the binary search aspect adds only a logarithmic number of calls to the decision problem, preserving polynomial-time complexity.

Problem 3.

Suppose you have a procedure which runs in polynomial time and tells you whether or not a graph has a Rudrata path but does not return such a path. Show that you can use it to develop a polynomial-time algorithm for RUDRATA PATH (which returns the actual path, if it exists). Note that Rudrata path is same as Hamiltonian path.

Let $G = (V, E)$ be a graph that will be decided whether it has a Hamiltonian path using a procedure DECIDE-HAM-PATH(G) that runs in polynomial time. We are to build an algorithm FIND-HAM-PATH(G) that returns the Hamiltonian path if it exists.

The algorithm is as follows:

- Use DECIDE-HAM-PATH(G). If it returns false, then return an empty path, denoting that no Hamiltonian path exists.
- Initialize an empty path P and set $V' = V$.
- For each vertex $v \in V'$ that you consider adding to P:

  For each vertex $u \in V'$ adjacent to v that could be the next vertex in P:

  - Temporarily remove the edge (v, u) from E, resulting in a new graph $G' = (V, E \setminus \{(v, u)\})$.
  - If DECIDE-HAM-PATH(G') returns true then a Hamiltonian path still exists in G' without the edge (v, u), implying (v, u) is not critical for the path at this step.
  - If DECIDE-HAM-PATH(G') returns false then the edge (v, u) is necessary for the Hamiltonian path, so add u to P and permanently remove v from V'.
  - Restore the edge (v, u) if it was not added to P.
- The process is then repeated until all vertices are in P.

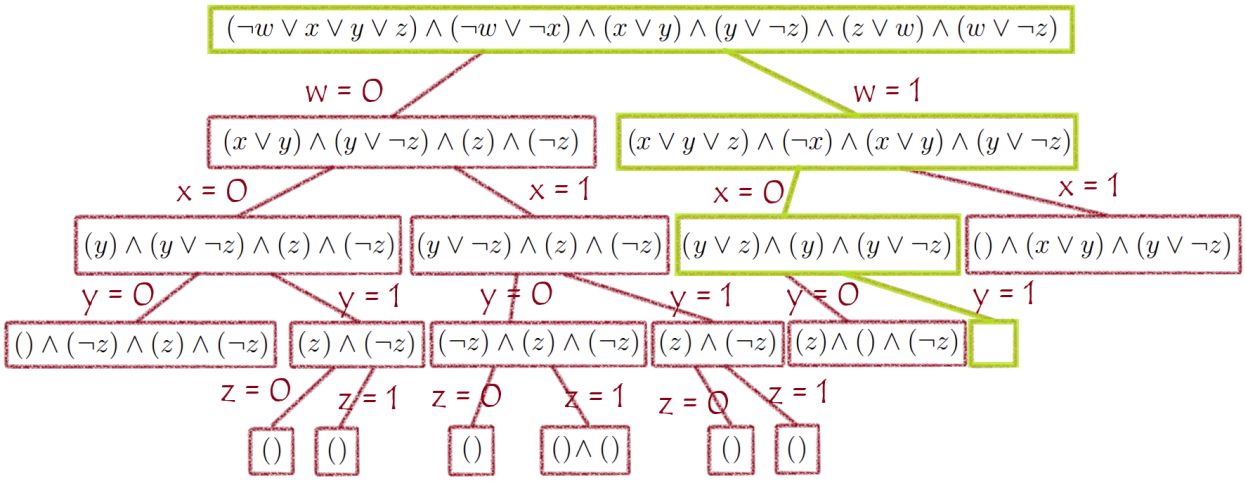
The algorithm ensures that at each step, the choice to add an edge to P is only made if removing the edge breaks the possibility of any Hamiltonian path. This means the added edge is part of the necessary path. Each edge removal and subsequent decision check involves a polynomial-time call to DECIDE-HAM-PATH. Since each edge is considered at most once, and each consideration involves a polynomial number of operations, the total time complexity remains polynomial.

Problem 4.

Draw the computation tree when doing backtracking for the formula

$$(\neg w \vee x \vee y \vee z) \wedge (\neg w \vee \neg x) \wedge (x \vee y) \wedge (y \vee \neg z) \wedge (z \vee w) \wedge (w \vee \neg z)$$

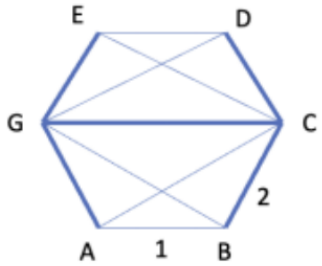
. Is ϕ satisfiable?



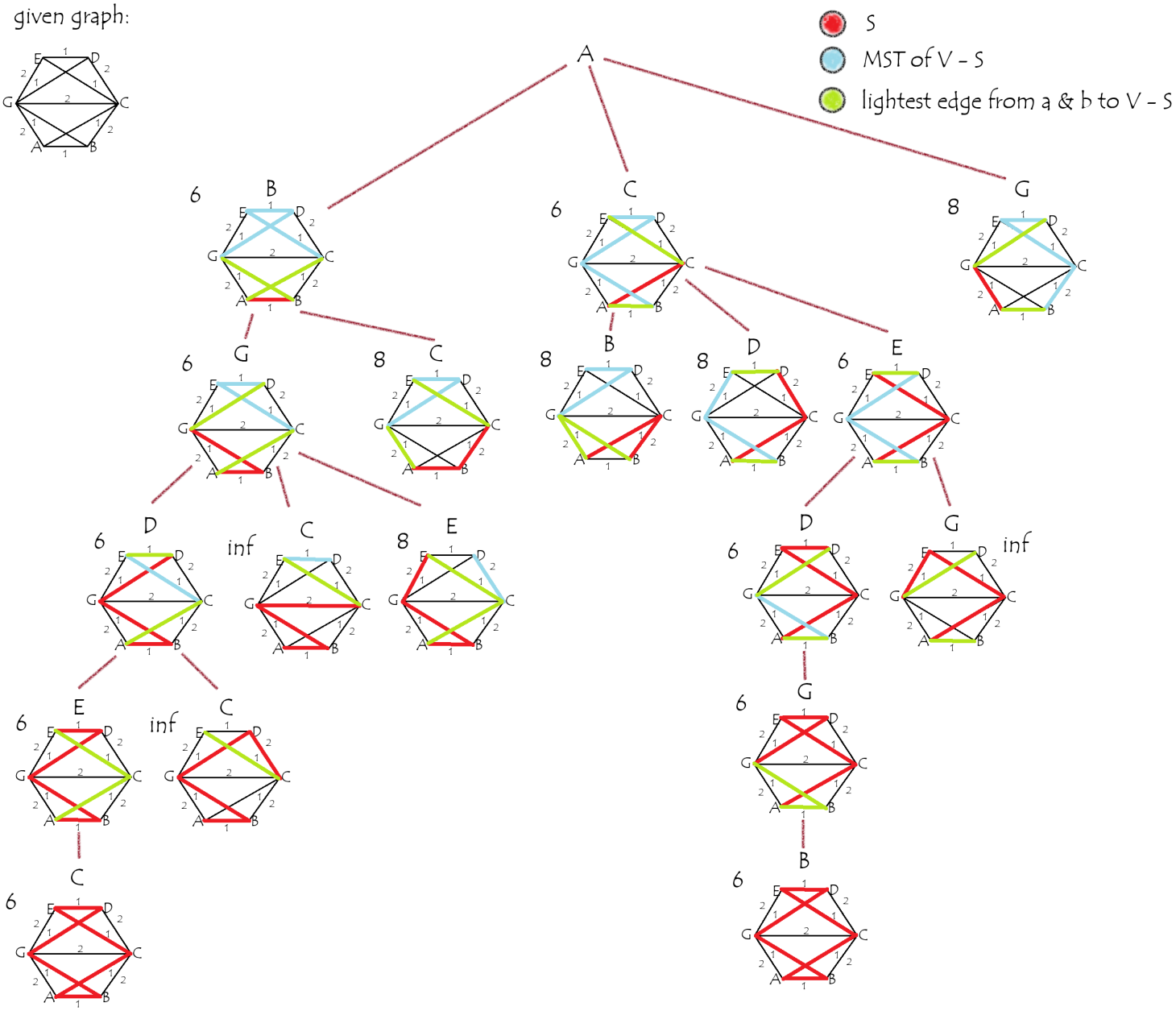
Based on the drawn computation tree, ϕ is satisfiable through $w = 1$, $x = 0$, and $y = 1$ edges. This implies that the solutions to ϕ are:

w	x	y	z
1	0	1	0
1	0	1	1

Problem 5.



Use branch and bound to find a solution for TSP for the above graph, where the thick lines have length 2 and the thin lines have length 1.



Based on the above branch and bound, the minimum cost for the TSP of the given graph is 6 with two possible paths being taken:

- $[A, B, G, D, E, C]$
- $[A, C, E, D, G, B]$