CPSC 320 2024S: Assignment 5

This assignment is due **Friday, August 8 at 7 PM**. Late submissions will not be accepted. Please follow these guidelines:

- Prepare your solution using LaTeXand submit a pdf file. Easiest will be to submit using the .tex file provided. For questions where you need to select a circle, you can simply change `\fillinMCmath` to `\fillinMCmathsoln` .
- Enclose each paragraph of your solution with `\begin{soln}Your solution here...\end{soln}`. Your solution will then appear in dark blue, making it a lot easier for TAs to find what you wrote.
- Clearly label your answer to each question.
- Submit the assignment via GradeScope. Your group must make a **single** submission via one group member's account, marking all other group members in that submission **using GradeScope's interface**.
- After uploading to Gradescope, link each question with the page of your pdf containing your solution.

Before we begin, a few notes on pseudocode throughout CPSC 320: Your pseudocode should communicate your algorithm clearly, concisely, correctly, and without irrelevant detail. Reasonable use of plain English is fine in such pseudocode. You should envision your audience as a capable CPSC 320 student unfamiliar with the problem you are solving. You may **neither** include what we consider to be irrelevant coding details **nor** assume that we understand the particular language you chose. (So, for example, do not write `#include <iostream>` at the start of your pseudocode, and avoid language-specific notation like C/C++/Java's ternary (question-mark-colon) operator.)

Remember also to **justify/explain your answers**. We understand that gauging how much justification to provide can be tricky. Inevitably, judgment is applied by both student and grader as to how much is enough, or too much, and it's frustrating for all concerned when judgments don't align. Justifications/explanations need not be long or formal, but should be clear and specific (referring back to lines of pseudocode, for example). Proofs should be a bit more formal.

On the plus side, if you choose an incorrect answer when selecting an option but your reasoning shows partial understanding, you might get more marks than if no justification is provided. And the effort you expend in writing down the justification will hopefully help you gain deeper understanding and may well help you converge to the right selection :).

Ok, time to get started...

Group Members

- Jason Mac: 98402373
- Jackson Chen: 38437752
- Irwin Wang: 91618181

Please list the CWLs of all group members here (even if you are submitting by yourself). We will deduct a mark if this is incorrect or missing.

1 Clustering 2.0

Recall the photo categorization problem described in Worksheets 5 and 6. An instance of the problem is given by

- n, the number of photos (numbered from 1 to n);
- E, a set of weighted edges, one for each pair of photos, where the weight is a similarity in the range between 0 and 1 (the higher the weight, the more similar the photos); and
- c, the desired the number of categories, where $1 \leq c \leq n$.

A solution to this problem is a *categorization*, which we define as a partition of photos into c (non-empty) sets, or categories. In the version of this problem we dealt with in class, our goal was to find the categorization that minimized the maximum inter-category edge weight. Suppose now that we change the goal of the photo categorization to **maximize the minimum intra-category edge similarity**. We call this the *Max-Min Clustering Problem* (MMCP).

1. [2 points] In class, we saw that an optimal greedy algorithm for minimizing the maximum inter-category edge weight was to initialize all photos in their own category and merge the photos adjacent to the highest-weight intercategory edge until we achieved the desired number of categories. Below is a greedy algorithm for MMCP, which starts with all photos in a single category and then separates two photos that are joined by a lowest-weight intracategory edge. This is repeated until we achieve the desired number of categories.

```
function MMCP-Greedy(n, E, c)
▷ n ≥ 1 is the number of photos
▷ E is a set of edges of the form (p, p′, s), where s is the similarity of p and p′
▷ c is the number of categories, 1 ≤ c ≤ n
    create a list of the edges of E, in increasing order by similarity
    let C be the categorization with all photos in one category
    Num-C ← 1                ▷ Initial number of categories
    while Num-C < c do
        remove the lowest-similarity edge (p, p′, s) from the list
        if p and p′ are in the same category S of C then
            ▷ split S into two new categories T and T′ as follows:
            put p in T
            put p′ in T′
            for each remaining p″ in S do
                put p″ in T if p″ is more similar to p than p′
                put p″ in T′ otherwise
            end for
            ▷ now S is replaced by T and T′ in C
```

Num-$\mathcal{C} \leftarrow$ Num-$\mathcal{C} + 1$
end if
end while
return $\mathcal{C}$
end function

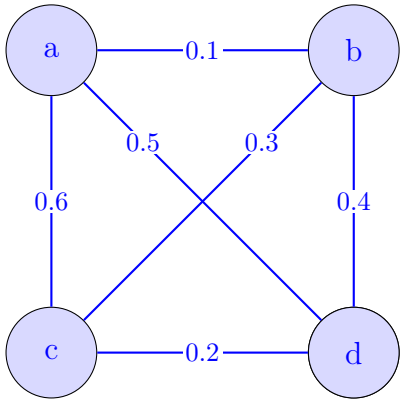
Give and explain a 4-node counterexample to show that this greedy algorithm is not optimal (i.e., does not always produce a solution that maximizes the minimum intra-category edge weight).

Let $C = C_1, C_2, \ldots, C_c$ be a partition of a nodes of a set V into c categories.

An edge, $(u, v) \in E$, is $intra - category$ if $\exists C_i \in C$ s.t. $u, v \in C_i$.

An edge, $(u, v) \in E$, is $inter - category$ if $\exists C_i, C_j \in C, i \neq j$ s.t. $u \in C_i$ and $v \in C_j$.

Consider the node structure, and we want to split this into two categories.



Following the algorithm we get the following.

$S = \{a, b, c, d\}$	one total category initially
$T = \{a\}, T' = \{b\}$	since (a, b) have lowest edge weight
$T = \{a, c, d\}, T' = \{b\}$	since $(c, a), (d, a)$ have heigher edge weight than $(c, b), (d, b)$
$C = \{T, T'\}$	final collection of categories

Observe the min-weight intra-category edge of this collection comes from $(c, d) \in T$ with weight 0.2.

But the better solution is $T = \{a, c\}, T' = \{b, d\}$ since the min-weight edge for each, respectively is 0.6 and 0.4.

We see that $0.4 > 0.2$, thus the greedy algorithm fails on this four node instance when consdiering two category partion.

2. [2 points] We've stated MMCP as an optimization problem. Give the decision version of MMCP, and prove that MMCP is in NP.

Defintition: We say a problem is in NP if there exists a verifier that can check in polynomial time.

Decision Version: Does there exist a partition of V into c categories such that all intra-category edges have similarity at least k, where $0 \leq k \leq 1$?

Proof that MMCP is in NP: To show a problem is in NP, by definition, we must show that a proposed solution can be verified in polynomial time.

A valid certificate is a partition of V into c categories $C = \{C_1, C_2, \ldots, C_c\}$. Note that $|C| \leq n$.

The verifier checks whether the similarity of all intra-category edges is at least k.

Let $|V| = n$. Since the graph is complete, $|E| = \frac{n(n-1)}{2} \in O(n^2)$.

We provide the following Verification Algorithm:

```
1: procedure VERIFY(E, C, k)
2:     for (u, v, s) ∈ E do
3:         for i = 1 to c do
4:             if u ∈ C_i and v ∈ C_i then
5:                 if s < k then
6:                     return No
7:                 end if
8:             end if
9:         end for
10:    end for
11:    return Yes
12: end procedure
```

Analyzing Running Time:

- Outer loop over all edges: $O(n^2)$
- For each edge, checking all $c \leq n$ categories for membership of u, v: $O(n)$

Total running time: $O(n^2 \cdot n) = O(n^3)$

Thus, the verifier runs in polynomial time, so MMCP is in NP.

3. [4 points] We will now show that MMCP is NP-hard (which, together with the proof from 3.2 that MMCP is in NP, means that it is NP-complete). We do this by reducing a known NP-hard problem to it. For this problem, you must do a reduction from the graph colouring problem, which (as a decision problem) is: given a graph G and a bound k, can we colour the vertices of G using no more than k colours such that no two adjacent vertices share a colour? You may assume that $k \leq n$, where n is the number of vertices in G.

   For this part of the question, you do not need to prove the correctness of your reduction (that comes next), but you should explain the reasoning behind your reduction and why it runs in polynomial time.

   Let an instance of graph colouring be given with graph $G = (V, E)$ and bound k. With $|V| = n$.

   We constuct the following partition and edge set for an instance of $MMCP$.

   For each pair $u, v \in V$, if $(u, v) \in E$ add $(u, v, 0)$ to E', otherwise if $(u, v) \notin E$ add $(u, v, 1)$ to E'.

   Then we have n photos for each $v \in V$ with the weighted edge set E'.

   We want $c = k$ categories, and weight at least 1 amongst the intra-category edges. This completes the reduction.

   The goal is to force pairs of vertices that have an edge into different categories that becomes my coloring.

   So, if I give them weight 0 then MMCP should not want to put them into the same category since that would not be at least 1.

   Then this runs in polynomial time since the only thing to do is to make a copy of V and construct a new edge set E'.

   The copy of V is just $O(n)$ time, while making E' is $O(n^2)$ since we want a complete edge set for each pair of vertices.

   Hence, overall its polynomial.

4. [3 points] To prove your reduction correct, you must prove that the answer to the MMCP instance is YES if and only if the answer the answer to the graph colouring instance is YES. For this part of the question, prove that if the answer to the graph colouring instance is YES, then the answer to your reduced MMCP instance is YES.

   Claim: Given the reduction, Graph Colouring instance is YES $\iff$ MMCP instance is YES.

   Proof $\implies$: Suppose Graph Colouring instance is YES.

   Observe having weight at least 1 is the same as being 1, since similarity is between 0 and 1.

   Then G can be coloured with no more than k colours, such that no adjacent vertices share a colour.

   WLOG assume G uses k colours, if it it used less, we could partition some colour into more colours.

   Let $f : V \to \{1, 2, \ldots, k\}$ be the colouring function such that if $(u, v) \in E$ then $f(u) \neq f(v)$.

   We show there exists partition of V to k categories so that the intra-category edges have weight 1.

   For each $j = 1, 2, \ldots k$ construct categories, $C_j = \{v \in V : f(v) = j\}$.

   Now consider an arbitrary category C_i and let $u, v \in C_i$.

   Then we have that $f(u) = f(v) = i$. By contrapostion, this means that $(u, v) \notin E$.

   By construction, we have $(u, v, 1) \in E'$. Since i was arbitrary, then every intra-category edge is weighted at least 1.

   This completes the proof of this direction.

5. [3 points] Now for the other direction of the if-and-only-if: prove that if the answer to the reduced MMCP instance is YES, then the answer to the original graph colouring instance is also YES.

   Proof $\impliedby$:

   Assume that MMCP instance is YES.

   Then we have n photoes partitioned into k categories with all intra-category edges weight at least 1.

   We denote the categories by $C_1, C_2, \ldots, C_k$.

   Then define the function $f : V \to \{1, 2, \ldots, k\}$ by

$$f(v) = \begin{cases} 1 & \text{if } v \in C_1 \\ 2 & \text{if } v \in C_2 \\ \vdots & \\ k & \text{if } v \in C_k \end{cases}$$

   It suffices to show that if $f(v) = f(u)$ then $(u, v) \notin E$.

   Suppose that $f(v) = f(u) = i$. Then $u, v \in C_i$ for some i with $1 \leq i \leq k$.

   Since, MMCP is YES, this means $(u, v, 1) \in E'$ as all intra-category edges are weighted at least 1.

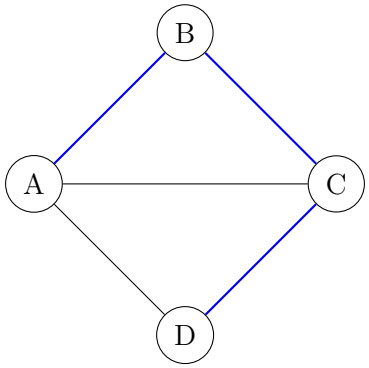
   But we only added $(u, v, 1)$ to E' iff $(u, v) \notin E$. Thus, $(u, v) \notin E$ and the graph G is k colourable.

   This completes the proof.

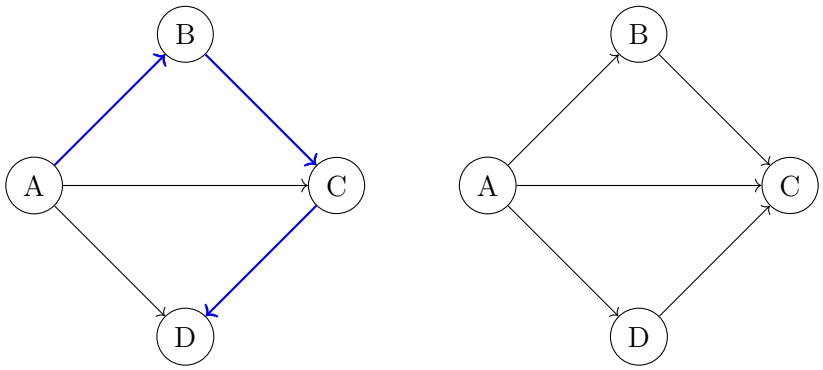
2 The Path To Victory 2.0

A Hamiltonian Path is a path in a graph that visits each vertex exactly once. In this question, we consider the variant of the Hamiltonian Path problem with the start and end node specified: that is, given a graph $G = (V, E)$ and two nodes s and t, does there exist a path from s to t that visits every node in G exactly once? You can may assume that G contains at least three vertices.

The Hamiltonian Path problem can be applied to graphs that are either undirected or directed. For example, the undirected graph below has a Hamiltonian Path from A to D given by (A, B, C, D), shown in blue:



For the directed graphs below, the graph on the left has a Hamiltonian Path from A to D, while the graph on the right does not have a Hamiltonian Path from A to D.



We refer to the undirected version of this problem as **UHP** and the directed version as **DHP**.

1. [3 points] Give a reduction from UHP to DHP.

   Let $G = (V, E)$ be an undirected graph. We construct a graph $G' = (V', E')$ by the following.

   We make a copy of V and set $V' = V$. Then for each $\{u, v\} \in E$ add $(u, v), (v, u)$ to E'.

   Then for $s, t \in V$ we ask DHP if there exists a Hamiltonian path from s to t in V' of G'.

2. [4 points] Prove the correctness of your reduction from UHP to DHP. That is, prove that the answer to your reduced DHP instance is YES if and only if the answer to UHP is YES.

   We'll say that $V = \{v_1, v_2, \ldots, v_n\}$ s.t. $v_1 = s$ and $v_n = t$. We ask for Hamiltonian path from s to t.

   ($\implies$): Suppose that the answer to the reduced DHP instance is YES.

   Then there is a Hamiltonian path from $s - t$ in $G' = (V', E')$. Denote the hamiltonian path by P.

   So, $P = (v_1, v_2, \ldots, v_{n-1}, v_n)$ and $(v_i, v_{i+1}) \in E'$ for $i = 1, 2, \ldots, n-1$.

   By construction, we have that $\{v_i, v_{i+1}\} \in E$ for $i = 1, 2, \ldots, n-1$.

   By definition, $(v_1, v_2, \ldots, v_{n-1}, v_n)$ is hamiltonian path in $G = (V, E)$.

   ($\impliedby$): Suppose that the answer to UHP is YES for the graph $G = (V, E)$.

   Denote the Hamiltonian path by $P = (v_1, v_2, \ldots, v_{n-1}, v_n)$ so that $\{v_i, v_{i+1}\} \in E$ for $i = 1, 2, \ldots, n-1$.

   Then since $\{v_i, v_{i+1}\} \in E$ we have that $(v_i, v_{i+1}) \in E'$, by construction.

   Then P is hamiltonian in $G' = (V', E')$.

   This completes the proof.

3. [7 points] Give a reduction from DHP to UHP. You **do not** need to prove the correctness of this reduction, but you should clearly explain the key components of your reduction and why they are there.

Let I be an instance of DHP. Denote the directed graph by $G = (V, E)$.

Write $V = \{s, v_1, v_2, \ldots, v_{n-2}, t\}$ so that $|V| = n$. We now create an instance I' of UHP.

For ease of notation, $(u, v)'$ will be an undirected edge, in particular, $(u, v)' = (v, u)' = \{u, v\}$.

We will construct a new undirected graph $G' = (V', E')$ for an instance I' of UHP.

For each $v \in V$ we construct new nodes v_{out} and v_{in}, and add their undirected edge to E'.

Then for all $(u, v) \in E$ add the edge $(u_{out}, v_{in})'$ to E', unless $u = t$ or $v = s$.

We are making a 'gadget' that encodes the directed nature of G, by routing paths through v_{out} and v_{in}.

The idea is that there is a ham path from s to t if and only if there is a ham path from s_{in} to t_{out}.

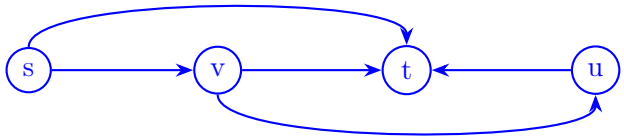
We force the instance I' to start at s_{in}, since it has no nodes 'incoming', it has to route through s_{out}.

Similarly, to get to t_{out} it must route through t_{in}, since t_{out} has no 'outgoing' nodes it connects to.

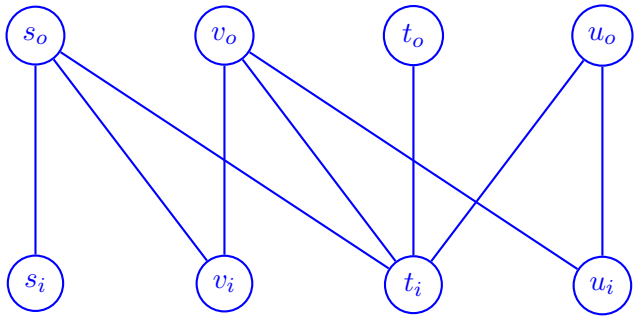
Lastly, we ask to UHP, does there exist a hampath from s_{in} to t_{out} in G'? This completes the reduction.

As an illustration,

Original Directed Graph



New Undirected Graph



To explain the ideas of the components, a sketch proof is provided that uses them.

Assume that I says yes. Let $P = (v^1, v^2, \ldots, v^n)$ be the hamiltonian path in G with $v^1 = s$ and $v^n = t$.

For each v^i, v^{i+1}, we have $(v^i, v^{i+1}) \in E$ s.t. $1 \leq i \leq n-1$. E' naturally encodes this hamiltonian path.

In other words, by construction, $(v^i_{out}, v^{i+1}_{in})' \in E'$, and we know an in node connects to its out node.

Thus, a hamiltonian path in G' would be $P' = (v^1_{in}, v^1_{out}, v^2_{in}, v^2_{out}, \ldots, v^n_{in}, v^n_{out})$.

The other direction uses the ideas we developed earlier. Assume that I says no to hampath question.

Then for every sequence $P = (v^1, v^2, \ldots, v^n)$ either some v^i is repeated or some $(v^i, v^{i+1}) \notin E$.

The idea now is that we can encode this sequence in G' to $P' = (v^1_{in}, v^1_{out}, v^2_{in}, v^2_{out}, \ldots, v^n_{in}, v^n_{out})$.

Hence, if some v^i is repeated, then some v^i_{in}, v^i_{out} is repeated or if $(v^i, v^{i+1}) \notin E$ then $v^i_{out}, v^{i+1}_{in} \notin E'$.

Thus, P' is not hamiltonian. Having taken arbitrary sequences, there is no hampath in G'.

The converted sequences we used in G are the only sequences of size $2n$ we need to consider in G'.

Thus, the reduction is loosely correct.

The proof that those sequences in G are the only ones that we need to consider in G' is non-trivial.

3 Very Special Problems

In this question, we consider special cases of NP-complete problems. Formally, we say that Problem B is a **special case** of Problem A if every instance of Problem B can be viewed as an instance of Problem A. We've seen examples of this in class: for example, 3-SAT is a special case of SAT (where every clause has length 3). Minimum spanning tree is a special case of Steiner Tree, in which the set of vertices we need to connect includes all the vertices in V.

1. [4 points] Consider the **Bounded-Leaf Spanning Tree Problem (BLST)**: given a graph $G = (V, E)$ and an integer k, does G have a spanning tree with no more than k leaves?

   Give an NP-complete problem that is a special case of BLST, and justify why this problem is a special case.

   Let an instance of BLST be given where k is specified to be 2 and that the answer is yes.

   Then we have some spanning tree T of G such that it has no more than 2 leaves.

   If G has at least 2 nodes, then T must have exactly 2 leaves, otherwise it means that G has one node.

   Since a spanning tree is connected, consider the path from one leaf to the other.

   Denote the leaf to leaf path by $P = (v_1, v_2, \ldots, v_n)$. Observe $(v_i, v_{i+1}) \in E$ for $i = 1, 2, \ldots, n-1$

   But by definition, this is precisely a hamiltonian path.

   Hence, hamiltonian path is a special case of BLST where $k = 2$.

2. [3 points] You showed in the previous question that there is an NP-complete problem that is a special case of BLST, and it is not difficult to show that BLST is in NP (though we are not asking you to do this). Does this imply that BLST is NP-complete? Justify your answer.

   This does not necessarily imply that BLST is NP-complete. We only so far know that BLST is NP.

   To prove BLST is NP-complete, we must show that for every other NP problem Y, that $Y \leq_p$ BLST.

   While we showed in the previous part that an NP-complete problem (HAMPATH) is a special case of BLST (for $k = 2$), this is not sufficient to conclude NP-completeness of BLST in general.

   For example, we can lax the conditions on HAMPATH and ask for any length path from $s - t$.

   The generalization becomes, does there exist a path from $s - t$? Disregaring needing exactly $|V|$ nodes.

   We know this is polynomial time solvable using BFS, assuming we have correct adjacency lists.

   Thus, the special case becomes NP-complete, but the general is not. Hence, the issue is lack of reduction.

   In particular, it remains to show that $BLST(k = 2) \leq_p BLST$ or equivalently $HAMPATH \leq_p BLST$.

   That is, we must show a polynomial time reduction from HAMPATH, not just observe that one is a special case of the other.

   Hence, unless such a reduction is explicitly given with proof of correctness, we cannot conclude that BLST is NP-complete.

3. [4 points] Give an example of a polytime-solvable problem you have seen in this class that is a special case of an NP-complete problem. Justify your answer.

Recall the problem we saw in tutorial 1, where we wish to know if a graph is bipartite.

A graph $G = (V, E)$ is **bipartite** if we can partition the vertices V into two disjoint sets U and W such that no two vertices in U are connected, and no two vertices in W are connected.

Then this problem is indeed a special case of the k-colouring problem with $k = 2$.

We prove this by showing that G is bipartite if and only if it is 2-colourable.

Suppose that G is 2-colourable. Then there is some function

$$f : V \to \{1, 2\}$$

such that if $(v, u) \in E$, then $f(u) \neq f(v)$.

Define the sets

$$V_1 = \{v \in V : f(v) = 1\}, \quad V_2 = \{v \in V : f(v) = 2\}.$$

We see $V_1 \cap V_2 = \varnothing$, otherwise there exists v such that $f(v) = 1$ and $f(v) = 2$, a contradiction.

Now, let $(u, v) \in V_1$. Then $f(u) = f(v) = 1$, which implies by contrapositive that $(u, v) \notin E$.

A similar statement holds for V_2, and thus G is bipartite.

Now, suppose G is bipartite. Then there is $V_1, V_2 \subseteq V$ so that no two vertices in either set are connected.

Consider the function

$$f : V \to \{1, 2\} \quad \text{defined by} \quad f(v) = \begin{cases} 1 & \text{if } v \in V_1, \\ 2 & \text{if } v \in V_2. \end{cases}$$

Let $(u, v) \in E$. Since G is bipartite, u and v cannot both be in the same set V_1 or V_2.

Hence, $f(u) \neq f(v)$, and thus G is 2-colourable.

Now consider running a BFS on a graph G starting from some node v, and colour it 1.

Then for each traversal step at a vertex v, for every edge (v, u) we travel along, colour u with the opposite colour of v if it is not coloured yet.

If u has already been coloured and its colour is the same as v, then report that G is not bipartite.

This algorithm runs in $O(n + m)$ time, since we only add the extra computation of assigning colours to each vertex during the BFS traversal.