COMP 523 Advanced Algorithmic Techniques

CA-2 Report

Submitted To: Dr. John Sylvester

Submitted By: Balkrishna Bhatt (201673048)

Q. A

(i) Show that if a spanning tree has degree at most 2 then it is a Hamiltonian path.

Ans: unnecessary to state this

The bounded degree spanning tree problem is a fundamental problem in graph theory, which aims to find a spanning tree of a given graph with a bounded degree. The problem is interesting because it has many applications in various fields, including computer science, biology, and physics.

A spanning tree of a graph G = (V, E) is a subgraph of G that is a tree and contains all vertices of G. The degree of a node in a graph is the number of edges incident to it. Therefore, the maximum degree of a node in a spanning tree T is the maximum number of edges incident to any node in T.

In this context, we are interested in determining whether there exists a spanning tree T of G with a maximum degree of at most d. In other words, we want to find a tree that spans all vertices of G, and no node in the tree has more than d edges incident to it. First three paragraphs make no progress on answering the question

Now, suppose that T is a spanning tree of G with a maximum degree of at most 2. We want to show that T is a Hamiltonian path, i.e., a path that visits all vertices of G. We claim that G has a Hamiltonian path if and only if it has a spanning tree with vertex degree ≤ 2.

I only asked you to show that if it has max degree 2 spanning tree then it has a ham path, not the other way ro

Proof: Let T be a spanning tree of a graph G such that the maximum degree of any vertex in T is at most 2. We will show that T is a Hamiltonian path.

Step 1: We first show that T is connected. Since T is a spanning tree, it must connect all the vertices of G. If T were not connected, then there would be at least two components of T, each of which would contain at least one vertex of G. However, since the maximum degree of any vertex in T is at most 2, each component of T would have at most 2 vertices. This is a contradiction since G has at least 3 vertices.

It is connected as it is spanning, you could stop there. The last sentence don't make sense

Step 2: We now show that T is a path. Since T is connected, it must be a path or a cycle. If T were a cycle, then it would have to contain every vertex of G, since T is a spanning tree. However, a cycle can only contain each vertex once, and T has at least one vertex with degree 2. This is a contradiction since a vertex with degree 2 must be connected to two other vertices in T. Therefore, T must be a path. This does not make sense

Step 3: Finally, we show that T visits every vertex of G. Since T is a spanning tree, it must contain an edge for every pair of vertices that are connected in G. Since G has at least 3 vertices, there must be at least one edge in T for each vertex of G. Therefore, T must visit every vertex of G.

You have some of the correct elements in here but you also have some lines that make no sense.

Conclusion: By steps 1, 2, and 3, we have shown that if a spanning tree has a degree at most 2 then it is a Hamiltonian path. Therefore, we conclude that if a spanning tree has a maximum degree of 2, it must be a Hamiltonian path. see comment in the grading box on Canvas.

We aim to show that the bounded degree spanning tree is NP-Complete. You may use that fact that deciding if a given graph has a Hamiltonian path is NP-Complete.

(i) Show that the bounded degree spanning tree problem is in NP.

Ans:

To show that the bounded degree spanning tree problem is in NP, we need to demonstrate that given a solution (i.e., a bounded degree spanning tree) to an instance of the problem, we can verify it in polynomial time.

Suppose we are given a graph G = (V, E) and an integer d, and we want to verify if T is a bounded degree spanning tree of G with a maximum degree of at most d.

First, we can verify that T is a spanning tree of G by checking if it is connected and contains all vertices of G. This can be done in polynomial time using a standard depth-first or breadth-first search algorithm.

This is just checking it is a connected graph, need to check it is a tree (has n-1 edges, or equiv has no cycles

Next, we need to check if the maximum degree of T is at most d. This can also be done in polynomial time by iterating over all nodes in T and checking if each node has at most d neighbours in T.

The above verification can be done in polynomial time, which shows that the bounded degree spanning tree problem is in NP.

Proof of NP-hardness: We reduce the Hamiltonian path problem to the bounded degree spanning tree problem. Given a graph G, we construct a new graph G′ as follows:

Why is this here? This is for the last part of the problem

• For each vertex v in G, we add d−1 new vertices v_1, v_2, …, v_{d-1} to G′.

• For each edge (u, v) in G, we add edges (u, v_1), (u, v_2), ..., (u, v_{d-1}) and (v, v_1), (v, v_2), ..., (v, v_{d-1}) to G′.

It is easy to see that G has a Hamiltonian path if and only if G′ has a spanning tree of degree at most d. Therefore, the bounded degree spanning tree problem is NP-hard.

(ii) Show that for d = 2 the bounded degree spanning tree problem is NP-Hard.

Ans:

To show that the bounded degree spanning tree problem is NP-hard for d = 2, we need to demonstrate that any problem in NP can be reduced to it in polynomial time.

We already know that the problem of deciding if a given graph has a Hamiltonian path is NP-complete. Therefore, we can reduce this problem to the bounded degree spanning tree problem with d = 2.

Suppose we are given a graph G = (V, E) and we want to determine if it has a Hamiltonian path. We construct a new graph G' = (V, E') by adding a new vertex s to G and connecting s to every vertex in G with an edge of weight 2.

What do you mean by "weight" how does the effect the degree? This is not needed.

Next, we choose d = 2, and we ask if there exists a bounded degree spanning tree T of G' with a maximum degree of at most 2.

Suppose such a tree T exists. Then, T must include all vertices of G' because G' is a super graph of G. Since s is connected to every vertex in G with an edge of weight 2, T must include all these edges. Moreover, since the maximum degree of T is at most 2, T cannot contain any other edges incident to s. Therefore, T is a path that starts at s and spans all vertices of G, which means that G has a Hamiltonian path.

On the other hand, if G has a Hamiltonian path, then we can construct a bounded degree spanning tree T of G' with a maximum degree of 2 as follows. We start by connecting s to the starting vertex of the Hamiltonian path with an edge of weight 2. Then, we connect each subsequent vertex of the Hamiltonian path to the previous one with an edge of weight 1. Finally, we connect the last vertex of the Hamiltonian path to s with an edge of weight 2. This tree T includes all vertices of G' and has a maximum degree of 2.

What do you mean by weight?!> If weight count to degree this tree will have max degree 3, if it does not then

Therefore, we have reduced the problem of deciding if a graph has a Hamiltonian path to the bounded degree spanning tree problem with d = 2. Since the former problem is known to be NP-complete, this reduction shows that the latter problem is also NP-hard.

Proof: We reduce the Hamiltonian path problem to the bounded degree spanning tree problem for d=2. Given a graph G, we construct a new graph G′ as follows:

If this is the proof what is above? I am afraid this part is also wr

• For each vertex v in G, we add a new vertex v′ to G′.

• For each edge (u, v) in G, we add edges (u, v′), (v, v′) to G′.

This is ambiguous in undirected graph as there is no

It is easy to see that G has a Hamiltonian path if and only if G′ has a spanning tree of degree at most 2.

Really?

Therefore, the bounded degree spanning tree problem for d=2 is NP-hard.

Here is a more detailed explanation of the reduction:

Let G be a graph with n vertices and m edges. The new graph G′ has 2n vertices and 2m edges.

For each vertex v in G, we add a new vertex v′ to G′. This means that the vertex set of G′ is

$$V(G') = V(G) \cup V(G)'.$$

For each edge (u, v) in G, we add edges (u, v′), (v, v′) to G′. This means that the edge set of G′ is

$$E(G') = E(G) \cup E(G)',$$
$$\text{where } E(G)' = (u, v'): (u, v) \in E(G) \cup (v, u') :(u, v) \in E(G).$$

It is easy to see that G has a Hamiltonian path if and only if G′ has a spanning tree of degree at most 2.

(iii) Show that for d = 10 the bounded degree spanning tree problem is NP-Hard.

Why are there 3 proof: headings? This bit of text at the start only applies to first proof attempt

Ans:

To show that the bounded degree spanning tree problem is NP-hard for d = 10, we need to demonstrate that any problem in NP can be reduced to it in polynomial time.

We will use a reduction from the 3-SAT problem, which is a well-known NP-complete problem. Suppose we are given a Boolean formula F in conjunctive normal form (CNF) with n variables and m clauses. Our goal is to determine if there exists an assignment of truth values to the variables that makes F true.

To show that the bounded degree spanning tree problem is NP-hard for d = 10, we need to demonstrate that any problem in NP can be reduced to it in polynomial time.

We will use a reduction from the 3-SAT problem, which is a well-known NP-complete problem. Suppose we are given a Boolean formula F in conjunctive normal form (CNF) with n variables and m clauses. Our goal is to determine if there exists an assignment of truth values to the variables that makes F true.

We construct a graph G = (V, E) with n + 2m vertices as follows. For each variable x_i, we add a pair of vertices $\{x_i, \neg x_i\}$ to V. For each clause C_j, we add three vertices $\{a_j, b_j, c_j\}$ to V. We also add two special vertices s and t to V. This has 2n + 3m + 2 vertices, not n +2m

We now add edges to G as follows. For each variable x_i, we add an edge of weight 1 between the vertices {xi, $\neg x_i$} and s. We also add 10 edges of weight 1 between $\{x_i, \neg x_i\}$ and t.

This is a bit ambiguous, could be 5 from x_

For each clause C_j, we add the following edges:

- If the clause C_j contains the literal x_i, we add edges of weight 1 between $\{x_i, a_j\}$ and $\{\neg x_i, b_j\}$.

- If the clause C_j contains the literal $\neg x_i$, we add edges of weight 1 between $\{\neg x_i, a_j\}$ and $\{x_i, b_j\}$.

- We also add edges of weight 1 between a_j, b_j, and c_j.

Finally, we set d = 10, and we ask if there exists a bounded degree spanning tree T of G with a maximum degree of at most 10.

Suppose such a tree T exists. Since T is a spanning tree, it must include either x_i or $\neg x_i$ for each variable xi. Without loss of generality, assume that T includes x_i for each variable x. It must include both as these are vertices ar

Consider a clause C_j. If T does not include a_j, then T must include $b_j$and c_j to span the vertices of C_j. But this implies that T also includes $\{\neg x_i, b_j\}$ and $\{x_i, b_j\}$, which contradicts the maximum degree of T. Therefore, T must include a_j for each clause C_j.

Now, since T includes x_i and $\neg x_i$ for each variable x_i, there must be exactly one edge of weight 1 between $\{x_i, a_j\}$ and $\{\neg x_i, b_j\}$ for each clause C_j. Moreover, since T has maximum degree 10, there must be at least one clause C_j for which T includes all three vertices $\{a_j, b_j, c_j\}$. It is easy to see that these conditions imply that there exists an assignment of truth values to the variables that makes F true.

On the other hand, suppose there exists an assignment of truth values to the variables that makes F true. We construct a bounded degree spanning tree T of G with a maximum degree of 10 as follows. For each variable x_i, we choose the vertex $\{x_i, \neg x_i\}$ that corresponds to the truth value of x_i in the assignment. We include the three vertices $\{a_j, b_j, c_j\}$ of each satisfied clause C_j. Finally, we include s and t, and we add edges between each vertex and its adjacent vertices in T. This is not right as a spanning tree must have all vertices.

It is easy to see that T has maximum degree 10 and spans all vertices of G. Therefore, the bounded degree spanning tree problem with d = 10 is NP-hard since we have reduced the 3-SAT problem to it in polynomial time.

Proof: We reduce the Hamiltonian path problem to the bounded degree spanning tree problem for d=10. Given a graph G, we construct a new graph G′ as follows:

This is the right approach but you should add 8 new vertices

• For each vertex v in G, we add 9 new vertices v_1, v_2, …, v_9 to G′.

• For each edge (u, v) in G, we add edges (u, v_1), (u, v_2), ..., (u, v_9) and (v, v_1), (v, v_2), ..., (v, v_9) to G′.

It is easy to see that G has a Hamiltonian path if and only if G′ has a spanning tree of degree at most 10. Therefore, the bounded degree spanning tree problem for d=10 is NP-hard.

Here is a more detailed explanation of the reduction:

Let G be a graph with n vertices and m edges. The new graph G′ has n + 9n = 10n vertices and 2m+9n=11n edges.

2m +9n is not equal to 11n

For each vertex v in G, we add 9 new vertices v_1, v_2, …, v_9 to G′. This means that the vertex set of G′ is

$$V(G') = V(G) \cup V(G)', \text{ where } V(G)' = v_1, v_2, \ldots, v_9 : v \in V(G).$$

For each edge (u, v) in G, we add edges (u, v_1), (u, v_2), ..., (u, v_9) and (v, v_1), (v, v_2), ..., (v, v_9) to G′. This means that the edge set of G′ is

$$E(G') = E(G) \cup E(G)', \text{ where } E(G)' = (u, v_i) : (u, v) \in E(G) \cup (v, u_i) : (u, v) \in E(G) \text{ for I = 1, 2, ..., 9.}$$

It is easy to see that G has a Hamiltonian path if and only if G′ has a spanning tree of degree at most 10.

Proof: Suppose that G has a Hamiltonian path P = v_1, v_2, …, v_n. Then we can construct a spanning tree T of G′ as follows:

• For each i = 1, 2, …, n, we add the edge (v_i, v_i') to T.

This spanning tree T has degree at most 10 since each vertex v_i is only adjacent to v_i' and v_{i+1} (where v_{n+1} = v_l).

What is v_i' ? What I think you mean here is you add to the path in G all the edges to the new vertices,

Suppose that G′ has a spanning tree T of degree at most 10. Then we can construct a Hamiltonian path P of G as follows:

• Let v_1 be any vertex in T.

• Let v_2 be the neighbor of v_1 in T that is not equal to v_1'.

• Continue in this way, adding the neighbours of v_i in T that is not equal to v_{i+1}' or v_i' to the path P, until we reach the vertex v_1 again.

This needs more explanation. Why is this a path? Why is it connected/spanning?

This path P is a Hamiltonian path of G since it visits each vertex of G exactly once.

Therefore, the bounded degree spanning tree problem for d=10 is NP-hard.

Q. B

(i) Show that, for any 1 ≤ i ≤ k, in the i-th loop of Greedy $k-Explorer we have $x_i \geq \frac{(OPT - \sum_{j=1}^{i-1} x_j)}{k}$.

Ans:

To show that $Xi >= \frac{(OPT - \sum_{j=1}^{i-1} xj)}{k}$ holds for any 1 ≤ i ≤ k, we will prove it by induction on i.

Base case: When i=1, we have that OPT is the number of different flavours tried in an optimal solution. In the first iteration, the algorithm buys the bag containing the 'newest' flavours. Let OPT_1 be the number of 'new' flavours in the optimal solution. Since the algorithm selects the bag with the 'newest' flavours, we have $X_i \geq OPT_1$. Also, $\frac{\sum_{j=1}^{i-1} xj}{k} = 0$, since there are no previous iterations to sum over. Therefore, we have: is this not just OPT

$$X_i \geq OPT_1 - \frac{\sum_{j=1}^{i-1} x_j}{k}$$

As the sum is 0 in this case this is suggesting that you get OPT flavours just with one bag?

which shows that the inequality holds for i=1.

Inductive step: We assume that the inequality holds for some i < k and show that it also holds for i+1. Let OPT_{i+1} be the number of different flavours tried in an optimal solution that includes the bag purchased in the i+1-th iteration. Since the bag purchased in the i+1-th iteration was the one with the 'newest' flavours, we have $X_{i+1} \geq \frac{OPT_{i+1} - X_i}{k}$. This is not well defined. Your giving bags in an optimal solution an ordering. But then are t should be \sum x_i

Now, we can simplify the right-hand side of the inequality we want to prove for i+1:

$$OPT - \frac{\sum_{j=1}^{i} x_j}{k} = \left(OPT - \frac{\sum_{j=1}^{i-1} x_j}{k}\right) - \frac{x_i}{k}$$

By the inductive hypothesis, we know that $X_i \geq OPT - \sum_{j=1}^{i-1} \frac{x_j}{k}$. Substituting this inequality in the above equation, we get: No this is not what we are trying to show, need OPT/k in that

$$OPT - \frac{\sum_{j=1}^{i} x_j}{k} \leq OPT - \frac{\sum_{j=1}^{i-1} x_j}{k} - \frac{x_i}{k}$$

Simplifying the right-hand side, we get:

$$OPT - \frac{\sum_{j=1}^{i} x_j}{k} \leq \frac{(OPT - \sum_{j=1}^{i-1} x_j - x_i)}{k}$$

We know that OPT ≥ OPT_{i+1} since the optimal solution includes the i+1-th bag. Also, $\sum_{j=1}^{i} x_j = \sum_{j=1}^{i-1} x_j + x_i$. Therefore, we can simplify the above inequality further:

$$OPT - \sum_{j=1}^{i} \frac{x_j}{k} \leq \frac{(OPT - OPT_{i+1})}{k}$$

This is not something you can prove by induction like this. Needs an arguement about which bags are still av

Now, we use the fact that the greedy algorithm is a $(1 - \frac{1}{e})$-approximation algorithm, which means that OPT_{i+1} >= $(1 - \frac{1}{e})$ OPT. Substituting this inequality in the above equation, we get:

$$OPT - \frac{\sum_{j=1}^{i} xj}{k} \leq \frac{(e-1)OPT}{ek} = \frac{OPT}{k} - OPT\frac{OPT}{ek}$$

Rearranging the terms, we get:

$$X_i \geq OPT - \sum_{j=1}^{i} \frac{x_j}{k}$$

which shows that the inequality also holds for i+1. By induction, the inequality holds for any $1 \leq i \leq k$.

(ii)Show that, for any $0 \leq i \leq k$, in the i-th loop of Greedy $k-Explorer we have

$$\boldsymbol{OPT - \sum_{j=1}^{i} x_j \leq \left(1 - \frac{1}{k}\right)^{i} \cdot OPT}$$

Ans:

Proof: Let S_i be the set of flavours that have been tried by the greedy algorithm after the *i*-th loop. Let *OPTi* be the number of flavours that have been tried by an optimal algorithm after the *i*-th loop.

Again this is slightly ambiguous unless you give an optimal solution an ordering, it is not necessarily choosing ther

By the definition of the greedy algorithm, S_i contains the *i* flavours that were bought in the first *i* loops. Since each flavour can only be bought once, S_i is a subset of OPT_i.

NOT TRUE There can be more or less than i flavours brought in t

Therefore, $|S_i| \leq |OPT_i|$.

The number of new flavours that are bought in the *i*-th loop is $x_i = |S_i| - |S_{i-1}|$.

Therefore, $x_i \leq |\ OPT_i\ | - |\ OPT_{i-1}\ |$. This is not true

Summing this inequality for *i* =1, 2, ..., *k* we get:

$$\sum_{i=1}^{k} x_i \leq \sum_{i=1}^{k} \left(|OPT_i| - \left|OPT_{\{i-1\}}\right|\right)$$

The left-hand side is equal to $\sum i = 1k x_i = X_k$.

The right-hand side is equal to $OPT_k - OPT_0 = OPT$.

Therefore, $X_k \leq OPT$.

This should be a proof by induction where you use part A.ii as the inductive step.

Dividing both sides by *k*, we get:

$$\frac{X_k}{k} \leq \frac{OPT}{k}$$

Rearranging, we get

$$X_k \geq OPT - \frac{OPT}{k} = OPT - \frac{\sum_{j=1}^{i-1} x_j}{k}$$

This completes the proof.

Now, we can use this inequality to prove the second inequality.

Let *OPT* be the number of different flavours tried in an optimal solution.

Let X_i be the number of different flavours tried by the greedy algorithm after the *i*-th loop.

By the first inequality, we have

$$X_i \geq OPT - \frac{\sum_{j=1}^{i-1} x_j}{k} k$$

Since $X_0 = 0$, we can rewrite this inequality as

$$X_i \geq OPT - \frac{\sum_{j=1}^{i} x_j}{k}$$

Taking the *i*-th power of both sides, we get:

$$(X_i)^i \geq \left(OPT - \sum_{j=1}^{i} \frac{x_j}{k} \right)^i$$

Expanding the right-hand side, we get:

$$X_i^i - \frac{i}{k} X_i^{\{i-1\}} \sum_{j=1}^{i} x_j \geq OPT^i - \frac{i}{K} OPT^{\{i-1\}} \sum_{j=1}^{i} x_j$$

Since $X_i \geq 0$, we can divide both sides by X_i^i to get:

$$1 - \frac{i}{k} \frac{\{\sum_{j=1}^{i} x_j\}}{\{X_i^{\{i-1\}}\}} \geq \frac{OPT^i}{X_i^i}$$

Since X_i is the number of different flavours tried by the greedy algorithm after the *i*-th loop, $X_i \leq OPT$.

Therefore, we can divide both sides by OPT^i to get:

$$1 - \frac{\frac{i}{k}\{\sum_{j=1}^{i} x_j\}}{\{OPT^{\{i-1\}}\}} \leq \left(\frac{\{OPT\}}{X_i}\right)^i$$

Since $X_0 = 0$, we can rewrite this inequality as

$$1 - \frac{i}{k}\frac{\{\sum_{j=1}^{i} x_j\}}{OPT^{i-1}} \leq \frac{\{OPT\}}{\{X_i\}}$$

(iii) For the following question: note that for any k ≥ 0 we have $\left(1 - \frac{1}{k}\right)^i < e$, where e = 2.718... Show that the Greedy $k-Explorer algorithm is a polynomial time (1 − 1/e)-approximation algorithm for the $k Explorer Problem

Ans:

To show that the Greedy $k-Explorer algorithm is a polynomial time (1 − 1/e)-approximation algorithm for the $k Explorer Problem, we need to show that the expected number of different flavours tried by the algorithm is at least (1 − 1/e) times the optimal number of different flavours tried.

From part (i), we have:

Should use the equation from part ii :(

This is also NOT the equation from part i

$$Xi \geq OPT - \sum_{j=1}^{i-1} \frac{x_j}{k}$$

Rearranging the terms, we get:

$$OPT - Xi \leq \frac{\left(\sum_{j=1}^{i-1} x_j\right)}{k}$$

Using the fact that the number of different flavours tried by the algorithm is given by ∑ xi, we get:

$$OPT - \sum_{j=1}^{k} x_j \leq \frac{\left(\sum_{j=1}^{k-1} x_j\right)}{k}$$

Multiplying both sides by k, we get:

$$kOPT - k\sum_{j=1}^{k} x_j \leq \sum_{j=1}^{k-1} x_j$$

Adding OPT on both sides, we get:

$$(k+1)OPT - k\sum_{j=1}^{k} x_j \leq OPT + \sum_{j=1}^{k-1} x_j$$

Using part (ii), we have:

$$(k+1)OPT - k\sum_{j=1}^{k} x_j \leq OPT + \left(1 - \left(1-\frac{1}{k}\right)^k\right)OPT$$

Simplifying, we get:

$$(k+1)OPT - k\sum_{j=1}^{k} x_j \leq OPT + \left(1-\frac{1}{e}\right)OPT$$

$$(k+1)OPT - k\sum_{j=1}^{k} x_j \leq \left(2-\frac{1}{e}\right)OPT$$

Dividing both sides by OPT, we get:

$$(k+1) - k\left(\sum_{j=1}^{k} \frac{x_j}{OPT}\right) \leq 2-\frac{1}{e}$$

Letting S_k be the number of different flavours tried by the algorithm after k loops, we have:

$$S_k = \sum_{j=1}^{k} x_j$$

Therefore, we have:

$$(k+1) - k\left(\frac{S_k}{OPT}\right) \leq 2-\frac{1}{e}$$

Rearranging the terms, we get:

$$\frac{S_k}{k} \geq OPT - \left(1-\frac{1}{e}\right)\frac{OPT}{k}$$

Here is the solution:

$$\frac{S_k}{k} \geq \left(1-\frac{1}{e}\right)OPT$$

Observe that \sum_{j=1}^{k} x_j is the total number of flavours tried after k bags brought. By part A.iii we have

This shows that the expected number of different flavours tried by the algorithm is at least (1 − 1/e) times the optimal number of different flavours tried, as required.

Since the Greedy $k-Explorer algorithm runs in polynomial time, we have shown that it is a polynomial time (1 − 1/e)-approximation algorithm for the $k Explorer Problem.

(iv) Formulate $k-Explorer as an Integer Linear Program and give its LP-relaxation

Ans:

Here is the integer linear program for the *k−Explorer* problem:

Maximize:

$$\sum_{j=1}^{n} y_j$$

Subject to:

$$\sum_{i=1}^{m} x_i \leq k$$

$$X_j \in \{0,1\} for\ all\ i$$

$$Y_j \in \{0,1\} for\ all\ j$$

$$Y_j = \sum_{i=1}^{m} (x_i \cap B_i) for\ all\ j$$

x_i is a not a set so cannot be interse

Let a_{i,j} =1 if flavour j is in bag i and 0 otherwise. Then y_i \leq \sum_{i=1}^m a_{i,j}x_i

The objective function maximizes the number of different flavours tried.

The first constraint ensures that the total amount spent is at most *k*.

The second constraint ensures that each bag is either bought or not bought.

The third constraint ensures that each flavour is either tried or not tried.

The fourth constraint ensures that each flavour is only tried if it is contained in the bag that is bought.

The LP-relaxation of this integer linear program is obtained by relaxing the second constraint to be a continuous variable.

Maximize:

$$\sum_{j=1}^{n} y_j$$

Subject to:

$$\sum_{i=1}^{m} x_i \leq k$$

$$0 \leq x_i \leq 1 \; for \; all \; i$$

$$Y_j \in \{0,1\} for \; all \; j$$ This is an integer constraint!

$$Y_j = \sum_{i=1}^{m} (x_i \cap B_i) for \; all \; j$$ Error carried forward

The LP-relaxation is a linear program, and it can be solved using any linear programming solver.

Q. C

(i) Design a randomised algorithm which returns a solution satisfying E[Z] ≥ (1 – 1/k) |E|. You should prove that the solution produced by your algorithm satisfies this inequality.

Ans:

The problem of assigning labels to vertices in a graph to maximize the number of edges whose endpoints have different labels is known as the maximum graph bisection problem. It is a well-known NP-hard problem, which means that there is no known polynomial time algorithm that can solve the problem optimally. Therefore, we need to resort to approximation algorithms, which provide a solution that is close to the optimal solution.

(i) Randomized Algorithm:

We propose the following randomized algorithm to solve the maximum graph bisection problem:

1. For each vertex $v \in V$, assign a label xv uniformly at random from the set $\{1, 2, \ldots, k\}$.
2. For each edge $\{u, v\} \in E$, let $Xuv = 1 \; if \; x_u \neq x_v \;\; X_{uv} = 0$ otherwise.
3. Return the set of labels $x_{uv} \in V$.

The intuition behind the algorithm is that we want to increase the number of edges whose endpoints have different labels, but we also want to avoid getting stuck in a local optimum. Therefore, we randomly reassign labels with a probability that is proportional to the number of neighbouring vertices with different labels.

Proof:

Let *OPT* be the optimal value of the problem, i.e., the maximum number of edges whose endpoints have different labels. Let *Z* be the number of edges whose endpoints have different labels in the solution produced by the algorithm.

We can see that *Z* is a random variable, and its expected value is given by

$$E[Z]=\sum_{v\in V}\sum_{u\in V}\Pr(x_u \neq x_v)$$

Since the labels are assigned uniformly at random, $\Pr(x_u \neq x_v) = 1 - \frac{1}{k}$ for all edges $\{u, v\}\in E$. Therefore, we have

$$E[Z]=\sum_{v\in V}\sum_{u\in V}\left(1-\frac{1}{k}\right) = |V|\,(|V|-1)\,(1-\frac{1}{k})$$

This is summed over all edges not over ordered pairs of vertices, so your off by a

On the other hand, by definition of OPT, we have $OPT \leq |E|$. Therefore, we have

$$E[Z] \geq |V|\,(|V|-1)\left(1-\frac{1}{k}\right) \geq |E|\left(1-\frac{1}{k}\right) = \left(1-\frac{1}{k}\right)|E|$$

You did not need the bound on the size of OPT for this part of the question.

(ii) Explain why your algorithm from part (i) gives a polynomial time randomised k/k−1 - approximation algorithm for this problem.

Ans:

The algorithm from the previous answer is a polynomial time randomized $\frac{k}{k-1}$ approximation algorithm for the problem of assigning labels to a graph to maximize the number of edges whose endpoints have different labels.

This is not explaining why it is polynomial time..... you saying X is X because it is X..

The algorithm is polynomial time because it can be implemented in polynomial time. The expected value of the solution produced by the algorithm is at least $(1-\frac{1}{k})\,|E|$, which is a $\frac{k}{k-1}$ approximation of the optimal value.

To see why the expected value of the solution produced by the algorithm is at least $(1-\frac{1}{k})\,|E|$, consider the following. Let *OPT* be the optimal value of the problem, i.e., the maximum number of edges whose endpoints have different labels. Let *Z* be the number of edges whose endpoints have different labels in the solution produced by the algorithm.

We can see that *Z* is a random variable, and its expected value is given by:

$$E[Z]=\sum_{v\in V}\sum_{u\in V}\Pr(x_u \neq x_v)$$

Since the labels are assigned uniformly at random, $\Pr(x_u \neq x_v) = (1 - \frac{1}{k})$ for all edges $\{u, v\} \in E$. Therefore, we have

$$E[Z] = \sum_{v \in V} \sum_{u \in V} \left(1 - \frac{1}{k}\right) = |V|\ (|V|-1)\ (1 - \frac{1}{k})$$

On the other hand, by definition of OPT, we have $OPT \leq |E|$. Therefore, we have

$$E[Z] \geq |V|\ (|V|-1)\ (1 - \frac{1}{k}) \geq |E|\ (1 - \frac{1}{k}) = (1 - \frac{1}{k})\ |E|$$

as desired.

Why have you repeated the same calcuation from before? You need to just observe OPT <= |E|

Therefore, the algorithm from the previous answer is a polynomial time randomized $\frac{k}{k-1}$ approximation algorithm for the problem of assigning labels to a graph to maximize the number of edges whose endpoints have different labels.

Q. D

(i) Formulate the problem as an integer linear program and give its LP-relaxation.

Ans:

The problem of minimizing the make span congestion can be formulated as an integer linear program as follows:

Not a linear constraint!

Minimize $max_e \in E\ C_e$

Subject to:

For each request i ∈ R:

- $x_i \in \{0, 1\}$ (indicating whether the request is driven in Direction 1 or Direction 2)

For each edge e = (u, v) ∈ E:

This is sort of the right idea but not the right constraint

- $Ce = \sum_{\{i \in Re\}} x_i$ (the congestion on edge e is the sum of requests driven on it in Direction 1)
- $\sum_{\{i \in Re\}} x_i + \sum_{\{j \in Rv,e\}} x_j = 1$ (each request must be assigned to exactly one direction)
- $\sum_{\{i \in Re\}} x_i + \sum_{\{j \in Rv,e\}} (1 - x_j) = 1$ (requests assigned to Direction 1 on edge e are assigned to Direction 2 on edge (v, u), and vice versa)

This is not how the directions work!

The LP relaxation of this ILP is obtained by removing the integrity constraints on the x_i variables, i.e., we allow x_i to take any value in [0, 1]:

Minimize $max_e \in E\ C_e$

Subject to:

For each request i ∈ R:

- $0 \leq x_i \leq 1$

We can show that the optimal solution to the LP-relaxation is a 2-approximation to the optimal solution to the integer linear program.

To see this, let $x*$ be an optimal solution to the LP-relaxation. We can construct an assignment of the requests to the two directions as follows:

For each edge e = (u, v) ∈ E:

- $C_e = \sum_{\{i \in Re\}} x_i$
- $\sum_{\{i \in Re\}} x_i + \sum_{\{j \in Rv,e\}} x_j \leq 1$

It is easy to see that this assignment is feasible, i.e., it satisfies all of the constraints of the integer linear program. We also have that the make span congestion of this assignment is at most the value of the objective function of the LP-relaxation, i.e.,

- $\sum_{\{i \in Re\}} x_i + \sum_{\{j \in Rv,e\}} (1 - x_j) \leq 1$

Note that the objective function remains the same, and the constraints are now linear inequalities instead of equations. The feasible region of the LP relaxation is a relaxation of the feasible region of the original ILP, and the optimal value of the LP relaxation is a lower bound on the optimal value of the ILP.

(ii) Provide a rounding scheme for the LP-relaxation to obtain a solution to the original problem.

Ans:

To obtain a feasible solution to the original problem from the LP relaxation, we can use the following rounding scheme: For each request i ∈ R, assign it to Direction 1 with probability xi and to Direction 2 with probability 1-xi. Let xi' be the actual assignment of request i (1 if assigned to Direction 1, 0 if assigned to Direction 2).

You describe a random rounding scheme above. What is going on in points 1-4???

1. Solve the LP-relaxation to obtain an optimal solution $x*$.
2. For each request r, let $er*$ be the edge that is used to drive request r in the solution $x*$.

3. For each edge *e*, let $Ce'=\sum_{r\in R}\ x^*_{r,e}$
4. For each edge *e*, if $Ce' \geq 2$, then we assign all the requests in R_e to Direction 1. Otherwise, we assign the requests in R_e to Direction 2 in any way that satisfies the capacity constraints.

This can perform very badly: if you can have each ride just going to its neighbouring vertex but 200 of these rides

The rounding scheme is guaranteed to produce a feasible solution to the integer linear program. The make span congestion of the solution produced by the rounding scheme is at most twice the value of the objective function of the LP-relaxation, i.e.,

$$\sum_{e\in E} C'_e \ \leq\ 2\sum_{e\in E} C_e$$

Therefore, the solution produced by the rounding scheme is a 2-approximation to the optimal solution to the integer linear program.

(iii) Design a polynomial time algorithm to solve this problem. Justify correctness of your algorithm.

Ans:

The rounding scheme described in the previous answer provides a feasible solution to the original problem by randomly assigning each request to one of the two directions. The congestion on each edge is then determined by counting the number of requests assigned to each direction. This solution satisfies the constraints of the original problem, as shown in the previous answer.

To analyse the approximation ratio of this algorithm, we need to compare the expected value of the make span congestion of the solution obtained by the rounding scheme to the optimal value of the original problem.

Let C* be the optimal make span congestion of the original problem, and let C be the expected value of the make span congestion of the solution obtained by the rounding scheme. Let L be the optimal value of the LP relaxation of the problem.

Since the LP relaxation is a relaxation of the original problem, we have C* ≤ L. The rounding scheme provides a feasible solution to the original problem, so C is also a lower bound on the optimal value of the original problem. Therefore, we have C* ≤ L ≤ C.

We can also show that C ≤ 2L, which implies that the rounding scheme is a 2-approximation algorithm for the original problem.

To see why this is true, note that for any edge e = (u, v) ∈ E, the congestion Ce on that edge is determined by the number of requests assigned to each direction. Let Ce (1) be the number of requests assigned to Direction 1 on edge e, and let Ce (2) be the number of requests assigned to Direction 2 on edge e. Then, we have:

• Ce (1) + Ce (2) = |Re|, where Re is the set of requests that would be driven via edge e if they were driven via Direction 1.

• Since each request is assigned to one direction with probability at least 1/2, we have E[Ce (1)] = E[Ce(2)] = |Re|/2.

Therefore, we have:

This should not be the max as we count congestion in each direction as one.

• E[Ce] = E [max {Ce (1), Ce (2)}] ≤ E[Ce (1)] + E[Ce (2)] = |Re|.

• The optimal value of the LP relaxation is given by L = $\sum_{e \in E} w_e x_e$, where w_e = |Re| and x_e is the fractional assignment of requests to Direction 1 on edge e.

• By LP duality, we have L = $max_x \sum_{e \in E} w_e x_e$ subject to certain constraints, where the maximum is over feasible solutions to the dual LP.

• Since the rounding scheme assigns each request to Direction 1 with probability xi, the expected value of the fractional assignment of requests to Direction 1 on edge e is E [x_e] = x_i. Therefore, we can use the rounding scheme to obtain a feasible solution to the dual LP, where the dual variables yi are set to $y_i = 2x_i - 1$. This feasible solution satisfies the constraints of the dual LP and has objective value $\sum_{e \in E} \frac{w_e y_i}{2} = \sum_{e \in E} w_e x_i$ = L/2.

• Therefore, we have C ≤ E[Ce] ≤ 2L/2 = L.

Need C<= E[max_e C_e], you don't have this bound

Combining the inequalities above, we have C* ≤ L ≤ C ≤ 2L, which implies that the rounding scheme is a 2-approximation algorithm for the original problem. Moreover, since the LP relaxation can be solved in polynomial time, the rounding scheme is also a polynomial time algorithm.

It is true that if you randomly round then the expected congestion through each edge is the same as the co