1 Vertex cover

Let \( G(V, E) \) be a finite strongly-connected directed graph. Let \( w : E \to \mathbb{R}^+ \) be a positive weight function defined on the edges of \( E \). A vertex-disjoint cycle cover is a collection of simple circuits of length \( \geq 2 \), such that every vertex in \( V \) participates in exactly one such circuit. Prove that the minimum weight vertex-disjoint cycle cover can be found in a polynomial time.

Hint: it was briefly mentioned in the lecture that finding a maximum (weighted) matching in the bipartite graph requires polynomial time. You can use this fact without proving it.

Show that finding a perfect matching of a minimum weight in a weighted bipartite graph requires polynomial time as well. Think how a vertex-disjoint cycle cover in a general directed graph can be obtained from a perfect matching in a suitable bipartite undirected graph.

1.1 Minimum cost perfect matching TODO

You can find a minimum cost matching using a maximum cost matching algorithm by at first finding the largest cost \( c \) and fixing a constant \( C \geq c \) and then computing new weights \( w'(e) = C - w(e) \) for all edges. Previously minimum \( w(e) \) will be the maximum with the new \( w'(e) \) and in fact the relative order of all non-equal weight will be swapped.

The next question is how to ensure that the maximum matching is indeed a perfect matching. Assume that the bipartite graph has \( |A| = |B| = n \) vertices in both sets. Let \( m \) be the maximum cost of an edge in the maximum weight matching problem. Construct a new weight function \( w'' = w'(e) + n \cdot m \). This ensures that the perfect matching has weight \( > n \cdot n \cdot m \) whereas any matching with \( k < n \) edges has a smaller cost \( k(m + nm) \leq (n - 1)(m + nm) < n^2 m \). Therefore the maximum weight matching has to find a perfect matching.

1.2 Cycle cover idea

For a directed graph \( G(V, E) \) consider an undirected bipartite graph \( G'(V \cup V', E') \) where \( V' \) is a copy of \( V \) as follows. The bipartite graph has a weight function \( w' \). For each node \( v \in V \) there is one node \( v \) in the set \( V \) of the bipartite graph and a dual node \( v' \in V' \). For each directed edge \( e = (u,v) \in E \) there is an undirected edge \( e' = \{u,v\} \in E' \) in the bipartite graph. The weight \( w(e) = w'(e') \). In words, each directed edge \( (u,v) \) in \( G \) is represented as an undirected edge in \( G' \) where the set \( V \) corresponds to the starting vertices, in this case \( u \in V \) and the copies of the vertices in \( V' \) correspond to the end vertices of the edges, in this case \( v' \in V' \).
We can define the algorithm for finding a minimum cost cycle cover as in Algorithm 1.2.

**Algorithm 1** Algorithm for vertex-disjoint cycle cover

Input: Graph $G(V, E)$ with weight function $w$

Output: A minimum weight cycle cover $C$ for $G$

1: Build the bipartite graph $G'(V \cup V', E')$ from $G$
2: Find the minimum cost perfect matching $M$ for $G'$
3: Define the cycle cover $C = \{(u, v) : \{u, v'\} \in M, u \in V, v' \in V'\}$
4: return $C$

We have to show the one to one correspondence of the found perfect matching and the vertex-disjoint cycle cover. We show that there is a one to one correspondence for cost $k$ perfect matching and cycle cover. The condition for minimum cost perfect matching and vertex cover follows from this.

In addition, we have to show that this algorithm is polynomial time.

1.3 Cost $k$ perfect matching yields a cost $c$ vertex disjoint cycle cover

Assume that we have found a cost $c$ perfect matching $M$ for the described bipartite graph. By definition it uses $|V|$ edges. For every edge $\{u, v'\} \in M$ there is a directed edge $(u, v) \in E$ in the original graph $G$. We now consider the edges of the matching $M$ in the context of the original graph. Let $C$ be the set $M$ translated to graph $G$ as described by the algorithm.

For every vertex $v$ we had vertices $v$ and $v'$ in the bipartite graph $G'$. By definition, $M$ contains both vertices $v$ and $v'$, namely there exists some $u, w \in V$ such that $\{u, v'\} \in M$ and $\{v, w'\} \in M$ where $u$ and $w$ can be the same vertex. In the context of the original graph, this means that $v$ has one incoming $(u, v)$ and one outgoing edge $(v, w)$ in $C$. This holds for any vertex $v \in V$.

The two important questions are if $C$ is a cycle cover and if $C$ is vertex disjoint. We know that $C$ has to contain at least one cycle, because it has equal number of edges and vertices (it has one more edge than a tree with $|V|$ vertices could have). Assume, that it is not a cycle cover, therefore, there should exist some vertex $v_0$ such that we start from $v_0$ and follow the edges in $C$ but never reach $v_0$ again. This would hold for any $v_0$ not located on a cycle. Consider the path from $v$ as $v \rightarrow v_1 \rightarrow v_2 \rightarrow ...$ there are two possibilities:

1. We reach some vertex $v_k$ such that we can not make a next step. This is a contradiction, because by definition each vertex had one outgoing edge, therefore we could always take the outgoing edge from $v_k$.

2. At some reach some vertex $v_k \neq v_0$ that we have already visited on the path, assume that $v_k$ is the first such vertex. By definition this $v_k$ has to exist because we have a finite number of vertices and from the previous case we know that the path can not end. We must reach such vertex is less than $|V|$ steps.

Assume, that the previous vertex was $v_n$. Therefore, we must have edges $(v_n, v_k)$ that we took now and $(v_{k-1}, v_k)$ that we took the first time we visited the $k$'th element on the path. From the assumption that $v_k$ was the first such vertex we know that $v_n \neq v_{k-1}$, therefore we know
that \( v_k \) must have two incoming edges in \( C \). However, this is a contradiction, because then \( v'_k \)
should have been matched by two edges in \( M \) (edges \( \{v_n, v'_k\} \) and \( \{v_{k-1}, v'_k\} \) correspondingly).
This would invalidate the definition of a matching as a matching has to cover each vertex only once.

Therefore, each vertex \( v \) has to be located on a cycle, because we are able to reach it again by
following the edges in \( C \).

Finally, \( C \) is vertex disjoint, because each vertex had only one incoming and one outgoing edge,
but if several cycles shared a set of vertices then some vertices should have higher in our out degrees.

The cost of the vertex cover and the perfect matching is trivially the same \( c \), because by defi-
nition we use the same cost for the corresponding edges in the two problems and we use equivalent
edges in both constructions \( (C \) uses the same edges as \( M \).)

1.4 If there is a cost \( k \) cycle cover then there is a cost \( c \) perfect matching

In here, we look at the transformation in the other direction. Assume that we have a graph
\( G(V,E) \) and a vertex disjoint cycle cover \( C \) of cost \( c \) on that graph. Now we build the bipartite
graph \( G'(V \cup C', E') \) and transform the set \( C \) into a set \( M = \{\{u, v'\} : (u, v) \in C\} \). Obviously the
cost of \( M \) is \( c \) because we use the corresponding edges with the same costs as \( C \).

We need to show that \( M \) is in fact a perfect vertex cover. We have to show that each vertex
is matched exactly once. Note that both the matching \( M \) and \( C \) contain exactly \( |V| \) edges. So, it is actually only necessary to look at the case where some vertex \( a \in V \cup V' \) is matched more
than once. However, this is not possible as the corresponding original vertex \( v \) had exactly one
input and output edge in \( V \), therefore the input \( v' \) and output \( v \) components correspondingly are
connected to only one edge in \( M \). We must have either \( a = v \) or \( a = v' \). The fact that we have \( |V| \)
edges and \( 2|V| \) nodes where each has a matching degree at most 1 implies that all of the vertices
have to be used, because every edge has two endpoints.

1.5 Minimum cost vertex-disjoint cycle cover can be found in polynomial time

This is actually trivial, because building the bipartite graph can be done in time \( O(|V| + |E|) \) and
writing out the cycle cover \( C \) can be done in time \( O(|E|) \) because \( |E| \) is the upper bound for the
size of \( M \). By the analysis on the beginning of the task we can find the perfect matching with
minimum weight in polynomial time, therefore the total algorithm is polynomial.

2 Travelling salesman problem

Consider a finite complete directed graph \( G(V,E) \). Let \( w : E \to \{1, 2\} \) be a weight function
defined on the edges in \( E \). In other words, for every pair of vertices \( u, v \in V \), the weight of the
dge \( (u, v) \) is either 1 or 2.

1. Do the weights of the edges of \( G \) satisfy the triangle inequality? Prove or disprove.
2. Use the algorithm from Question 1 to give a factor $\frac{3}{2}$ approximation algorithm for the Traveling Salesman Problem for the graph $G$.

### 2.1 Triangle inequality

The triangle inequality holds if for all vertices $u, v, t \in V$ we have

$$w(u, v) \leq w(u, t) + w(t, v).$$

We can consider all the possible combinations of the weights that the triangles can have. We have to show that for all of these cases the triangle inequality is satisfied for all edges in the triangle.

1. $(1, 1, 1)$. The triangle inequality is clearly satisfied because $1 < 1 + 1$.
2. $(1, 1, 2)$. The triangle inequality is satisfied because $1 < 1 + 2$ and $2 \leq 1 + 1$.
3. $(1, 2, 2)$. The triangle inequality is satisfied because $1 < 2 + 2$ and $2 < 1 + 2$.
4. $(2, 2, 2)$. The triangle inequality is satisfied because $2 < 2 + 2$.

### 2.2 Algorithm for TSP

The algorithm described in Algorithm 2 uses the cycle cover of the graph $G$ and builds an Hamiltonian cycle from this cycle cover by arbitrarily connecting all cycles. Note that we can do that because the graph $G$ is complete. In addition, lets extend the weight function so that it can also be applied to a set of edges as $w(C) = \sum_{e \in C} w(e)$.

**Algorithm 2** Travelling salesman problem

**Input:** Graph $G(V, E)$ with weight function $w$

**Output:** A minimum weight TSP route $R$

1: Find a minimum cost cycle cover $C$ for $G$
2: Delete one edge from each cycle in $C$ to obtain $C'$
3: Connect the fragments from $C'$ into an Hamiltonian cycle $R$
4: return $R$

Step 3 of connecting the fragments $C'$ into $R$ can, for example, be done by fixing an ordering for $C'$ and then connecting the endpoints of paths in $C'$ along this ordering.

The approximation factor of this algorithm follows from the special conditions of the edge weights as they are always either 1 or 2. Lets denote the optimal TSP route length by opt. We know that the optimal route is also a valid cycle cover, therefore we know that

$$w(C) \leq \text{opt}$$

because the weight of $C$ has to be minimal among valid cycle covers. In addition, from the condition of the edge length we know

$$|V| \leq \text{opt} \leq 2|V|$$
because a cycle has $|V|$ edges with length either 1 or 2.

Let’s assume that $C$ contains $t$ cycles. Therefore

$$w(C') \leq w(C) - t$$

because then we delete $t$ edges where the weight of each edge is at least 1. In addition, we have

$$w(R) \leq w(C') + 2t$$

because we also have to add $t$ edges to build the final cycle by connecting $C'$, each edge that we have can have a maximum weight of 2.

To put this together we get the following

$$w(R) \leq w(C') + 2t \leq w(C) - t + 2t = w(C) + t \leq \text{opt} + t .$$

However, we can also estimate the size of $t$ because each cycle of $C$ has to contain at least 2 vertices and therefore we can have at most $\frac{|V|}{2}$ cycles. Therefore $t \leq \frac{|V|}{2} \leq \frac{\text{opt}}{2}$. When we put this together with the previous analysis we get

$$w(R) \leq \text{opt} + t \leq \text{opt} + \frac{\text{opt}}{2} = \frac{3}{2} \text{opt} .$$

The last line proves the approximation factor of the proposed algorithm for the travelling salesman problem as $R$ is the final route of the salesman and $w(R)$ is the length of this route. Therefore, the solution $R$ is always within $\frac{3}{2}$ factor of the optimal solution.

3 Moving company

The moving company has to load $n$ pieces of furniture $a_1, a_2, \ldots, a_n$ into $k > 1$ trucks. For each $i = 1, 2, \ldots, n$, the weight of $a_i$ is $w_i$ kilograms. Assume that each truck can carry unlimited load. The goal is to minimize the maximum total weight of the furniture loaded to each truck.

Consider the following greedy algorithm. The furniture pieces are ordered in an arbitrary order. The piece under consideration is placed into the truck with the smallest load. Show that at the end of the algorithm run, the weight of the load of the most heavily loaded truck is at most 2 times the optimum.

By the cost of the optimal solution here we mean the weight of the heaviest truck in this solution. Lets denote this by $\text{opt}$. In addition, denote $W = \sum w_i$ the total sum of the objects. Then we can calculate the average weight of a truck $\text{avg} = \frac{W}{k}$. In addition, we know that $\text{opt} \geq \text{avg}$ because the heaviest truck can either have average weight or weight above the average. In addition, the bag with the least weight while we still add items always has weight less than the average $\text{avg}$. Moreover, for each object $a_i$ the corresponding weight $w_i \leq \text{opt}$ because each item has to be in some truck and no items can decrease the weight of a truck.

Assume, by contradiction, that the algorithm finds a solution where the weight of the heaviest truck is $\text{cost} > 2 \cdot \text{opt}$. Consider the step where we add the last element $a_m$ to this truck. This
object has weight $w_m$. From the previous we know that when we chose this truck as a truck where be put $a_m$ it had the least weight among all trucks. Therefore, the weight \( \text{weight} = \text{cost} - w_m \) of this truck before adding $a_m$ follows the previous observation as \( \text{weight} \leq \text{avg} \). In addition, we know that by assumption \( \text{cost} = \text{weight} + w_m > 2 \cdot \text{opt} \).

However, we can use the following to estimate the total weight of the heaviest truck. From the previous observations we know that

\[
\text{weight} + w_m \leq \text{avg} + w_m \leq \text{opt} + w_m
\]

however, we can put this together with the contradictory assumption to get

\[
\text{opt} + w_m > 2 \cdot \text{opt}
\]

which results in a contradiction

\[
w_m > \text{opt}.
\]

This is a contradiction because we would have to put $a_m$ to some truck and this would then obtain larger weight than the heaviest truck in the optimal solution but this item $a_m$ has to be present in the optimal solution.

Therefore, it is not possible that this algorithm finds a solution that has worse approximation factor than 2, hence it definitely gives at least a factor 2 approximation.

4 Moving company with two trucks

In this problem, the company needs to load the $n$ pieces of furniture $a_1, a_2, \ldots, a_n$ into two trucks. As in the previous question, the weight of $f_i$ is $w_i$ kilograms, and each truck can carry unlimited load. The goal is to minimize the maximum total weight of the furniture loaded to each truck. Consider the same greedy algorithm. The furniture pieces are ordered in an arbitrary order. The piece under consideration is placed into the truck with the smallest load.

- Show that at the end of the algorithm run, the weight of the load of the most heavily loaded truck is at most $\frac{3}{2}$ times the optimum.

- Show an example of the weights $w_i, i = 1, 2, \ldots, n$, such that the output of the algorithm is exactly a factor $\frac{3}{2}$ of the optimum.

4.1 Approximation factor

The idea of this analysis is analogous to the previous task. The following observations still hold: By the cost of the optimal solution here we mean the weight of the heaviest truck in this solution. Lets denote this by $\text{opt}$. In addition, denote $W = \sum w_i$ the total sum of the objects. Then we can calculate the average weight of a truck \( \text{avg} = \frac{W}{2} \). In addition, we know that $\text{opt} \geq \text{avg}$ because the heaviest truck can either have average weight or weight above the average. In addition, the truck with the least weight while we still add items always has weight less than the average $\text{avg}$. Moreover,
for each object $a_i$ the corresponding weight $w_i \leq \text{opt}$ because each item has to be in some truck and no items can decrease the weight of a truck.

Consider the truck $b$ that is the heaviest among the two in the end of the algorithm. In addition, by contraction, assume that the total weight $\text{cost}$ of $b$ is larger than $\text{cost} > \frac{3}{2}\text{opt}$. Assume that $a_m$ is the last element that the algorithm placed to $b$. Then lets denote $\text{weight} = \text{cost} - w_m$ the weight of $b$ before adding $w_m$. At that point $b$ has to be lighter than the other truck $\bar{b}$. In addition, the total weight $\text{weight} \leq \frac{W-w_m}{2}$ because $W-w_m$ is the maximum total weight that these two trucks can have before adding $a_m$. As for the previous case with simple $\text{avg}$ the lighter truck also has to have less weight than the average weight that already is put to trucks.

We get the following

$$\text{cost} = \text{weight} + w_m \leq \frac{W-w_m}{2} + w_m = \text{avg} + \frac{w_m}{2} \leq \text{opt} + \frac{w_m}{2}.$$ 

If we put this together with the previous assumption that $\text{cost} > \frac{3}{2}\text{opt}$ we get the following

$$\frac{3}{2}\text{opt} < \text{opt} + \frac{w_m}{2},$$

where simplifying it yields

$$\frac{\text{opt}}{2} < \frac{w_m}{2}.$$ 

Therefore we have actually received a contradiction with one of the trivial initial observations as we now have

$$\text{opt} < w_m.$$ 

However, similarly to the previous case the optimal solution $\text{opt}$ has to contain $w_m$ in one of the trucks and we must have $w_i \leq \text{opt}$ for each $i$. Therefore, it is not possible that this algorithm finds a solution that is worse than $\frac{3}{2}\text{opt}$.

### 4.2 Example

We can also show that the approximation factor $\frac{3}{2}$ is strict for this problem. For example, consider a case with three elements $a_1, a_2$ and $a_3$ with corresponding weights $w_1 = 1, w_2 = 1$ and $w_3 = 2$. The obvious optimal solution has elements $a_1$ and $a_2$ in one truck and $a_3$ in the second truck. Hence, $\text{opt} = 2$.

However, if we consider the algorithm run with the order of elements as $a_1, a_2$ and finally $a_3$. The algorithm puts $a_1$ into one truck and $a_2$ into another (previously empty) truck. Then both trucks have weight 1 and $a_3$ is placed to one of these trucks. Therefore the cost of the solution from the algorithm is $\text{cost} = 1 + 2 = 3$. Clearly we now have $\text{cost} = \frac{3}{2}\text{opt} = \frac{3}{2} \cdot 2 = 3$. 

7