Algoritmekonstruksjon TDT4125 - Assignment 2

Ola Kristoffer Hoff

13th March, 2023

Task 1

In this task, we will deal with the weighted set cover problem and construct a primal-dual algorithm for it.

The weighted set cover problem is the problem in which we have a set of elements and a set consisting of subsets of these elements. Each subset has an associated cost and each element appears in at least one of the subsets. We want to select subsets such that the union of these, contain all the elements in the set and that the total cost of the selected subsets is as small as possible.

a Construct an integer program that describes the weighted set cover problem. Introduce the notation needed to describe the problem.

$$\min_x w^T x, \;\; s.t. \;\; Ax \geq b, \;\; x \in \{0,1\}^n$$

Here w is the weights such that the objective function becomes the minimisation of the weights that are active, given that x is either 1 or 0.

All values of x is either 1 or 0. Each row in A belong to a ground element e and the values in the rows are either a 1 or 0 denoting if a subset, S, denoted by the index of the column, contains the element e. b is a column vector of length n and all values are 1. We need at most n subsets S, since we have n elements $e \in E$, and in the worst case we could have one distinct element e in each S.

The constraint then states that all ground elements $e \in E$ must at least be in one subset $S_i, \; i \in I$. This then keeps the definition of the weighted set cover problem true.

b Formulate the dual of the LP relaxation of the integer program you constructed in a).

$$\max_y b^T y, \;\; s.t. \;\; A^T y \leq w, \;\; y \geq 0$$

We find the dual as done in the previous assignment, as the definition demands. Then we change the constraint from being either 1 or 0 to be greater than or equal to zero.

Here b have no effect other than summing y since it is just ones. Hence, we want to maximise the values of the dual y.

The constraint here is informally describes as follows: Each row of A^T is corresponding to a subset S and has values 1 or 0. This indicating if the given column, indexing an element $e \in E$, is in the subset. Hence, each row multiplied with y is a summation of the y's connected to the elements $e \in S_l$ which has to be less than or equal to the weight, w_l, of the subset.

c What are the complementary slackness conditions of the problem?

Complementary slackness states that we only have x_j equal to 1 in the primal integer program if the dual restriction j is tight.

In our case we have that if the primal-dual solution gives the index set I, given that our primal is an integer program we know that each $x_j = 1, \; \forall j \in I$. This guarantees that whenever $x > 0$ in the primal the corresponding dual constraint is tight, hence on these the complementary slackness is satisfied. If the case is that for all $y_i > 0$ the corresponding primal constraints are tight we would have a complete tight slackness, meaning we would have the optimal solution for the problem.

These are not as easily proven to be tight, since these are the values we change until the set cover is valid: $\bigcup S_j = E, \; j \in I$. We can however approximate them, cue the next task.

d What are the approximate complementary slackness conditions?

Continuing directly from the previous task; we can approximate the form of the complementary slackness condition. $\forall y_i > 0$ we have the equation[2]:

$$\sum_{j:e \in S_j} x_j = |\{j \in I : e_i \in S_j\}| \leq f$$

This equation informally states that the number of subsets S, or in other words, the size of I, is equal to the size of the set of the union of all subset $S_j,\ j \in I$.

I the case we can show that these complementary slackness conditions hold for a given α we have shown that the algorithm is of α-approximation.

e Describe a primal-dual algorithm to solve the problem.

Continuing the notation from the previous tasks we get the following algorithm as described in the compendium[2] (page. 120, Algorithm 7.1).

Algorithm 1 The prim-dual algorithm for the set cover problem.

$y \leftarrow 0$
$I \leftarrow \emptyset$
while $\exists e_i \notin \bigcup_{j \in I} S_j$ **do**
  Increase the dual variable y_i until there is some l such that $\sum_{j:e_j \in S_l} y_j = w_l$
  $I \leftarrow I \cup \{l\}$
return I

I will try to informally explain the algorithm. First we start with $y = 0$ which is a feasible dual solution given that all weights are: $w_j \geq 0$. The set of indices is empty so no conditions for the primal is held. Then we begin the loop; as long as there exists an element in the ground set, $e_i \in E$, that is not in any of the subsets, $S_j,\ j \in I$, we increase the dual variable y_i until a weight w_l is tight. Then we add l to the set I. When there exists no such e we return the set I as the solution to the set cover problem.

f What can you say about the constants in the approximate complementary slackness conditions after running the algorithm?

The compendium derives a proof for the case[2] (page. 120-121, Theorem 7.1):

$$\begin{aligned}
\sum_{j \in I} w_j &\leq f \cdot OPT \\
&\vdots \\
\sum_{j \in I} w_j &= \sum_{i=1}^{n} y_i \cdot |\{j \in I : e_i \in S_j\}| \\
|\{j \in I : e_i \in S_j\}| &\leq f \\
\sum_{j \in I} w_j &= \sum_{i=1}^{n} y_i \cdot f \\
&\vdots \\
\sum_{j:e_i \in S_j} x_j &= |\{j \in I : e_i \in S_j\}| \leq f
\end{aligned}$$

As we know x is a column vector of 1s and 0s indicating the "activation" of a subset, hence including it's weight. Thus, we see that f only needs to be larger or equal than the number of subsets we use. Hence, sine f bounds this value, we have that the cost is at most f times the dual solution.

g Use these constants to find an approximation factor for the algorithm.

This is answered in the previous task, how I structured the argument it was more natural to combine these questions.

Task 2

This task is inspired by a task in "The Design of Approximation Algorithms" (Williamson and Schmoys).

In the binary knapsack problem, we have n items, each with a weight $w_i > 0$ and a value $v_i > 0$. We can bring objects that together weigh up to a weight W and we want to maximize the value of the objects we bring.

a A naive greedy algorithm is to pick random items until we don't have room for more items. Does this algorithm provide a constant approximation factor? If so, what is it?

It does not. I will provide a simple counter example to show why. If we have a set where we have a item i that fills the weight alone, $w_i = W$, and has a value of one, $v_i = 1$. This could be picked at random. However, there could be $n-1$ elements with weight 1 and value P. If we assume $(n-1) = W$, so that we always can fit all other elements or element i we get:

$$\lim_{P \vee n \to \infty} \frac{v_i}{(n-1)P} = 0$$

As the number of other good items increase and/or the value of the items increase we see that the factor converges towards zero. Since the choice of the bad element is a valid choice in the random algorithm we can hence say that it does not have an approximation guarantee factor.

b Another possibility for a greedy algorithm is to start by sorting the items in descending order based on value per unit weight, v_i/w_i. Then you pick these items one by one, in order, until you cannot take the next item. If there is a single item that has a higher value than all these objects combined, the object (of all the objects) with the highest value is picked instead. Show that this algorithm has an approximation factor of 1/2.

Hint 1: Picking the objects with the best value to weight ratio provides an optimal solution for the fractional knapsack problem. Can you use this to show the approximation factor of this algorithm?

I think this is quite an elegant proof, and I will try to present it as such.

First we need to establish some trivial conditions and notation:

- I is the set of all n items.
- $w_i \leq W, \ \forall i \in I$, otherwise they would not fit alone as a solution.
- $\sum_{i=1}^{n} w_i > W$, otherwise the whole set would be the solution.

Since choosing the item with the highest value to weight ratio gives the optimal solution in the fractional version, it will here to, if it adds up in the integer range. Regardless we know that the integer solution is lower bound by the LP solution. If we do as the algorithm describes and pick an item until we can't fit the next item we reach the integer constraint of the problem. If we choose $k-1$ items choosing the k'th item would violate the constraint: $\sum_{i=1}^{k} w_i > W$. This k'th item is not necessarily the next item, it's the item of the remaining items with the highest value. The sum of the k items becomes an upper bound on the optimal solution. The max value of the first $k-1$ items and the k'th item will then be greater then the or equal to their average (half their sum). Since their sum was the upper bound on the optimal solution the max value must therefore be greater than half the optimum.

$$\frac{1}{2}OPT < \frac{\sum_{i=1}^{k} v_i}{2} \leq max\{v_k, \sum_{i=1}^{k-1} v_i\}$$

Hence, the algorithm has an approximation factor of $\frac{1}{2}$.

Task 3

This task is taken from "The Design of Approximation Algorithms" (Williamson and Schmoys).

In this task we look at the k-suppliers problem which is similar to the k-center problem. The input to the problem is a positive integer k, and a set of vertices V, along with distances d_{ij} between any two vertices i, j that obey the same properties as in the k-center problem. However in this variant of the problem, the vertices are divided into two sets. A subset $F \subseteq V$ of suppliers and a set $D = V \backslash F$ of clients. We want to select a subset $S \subseteq F$ of k suppliers so that the maximum distance from a client to the nearest supplier in S is minimum. In other words, we wish to find $S \subseteq F$, $|S| \leq k$, that minimizes $max_{j \in D} d(j, S)$
Note that as in the k-center problem, d_{ij} is a metric. That is, $d_{ij} \leq d_{ik} + d_{kj}$ for all i, j, $k \in V$.

a Find an approximation algorithm for the k-supplier problem with approximation factor of 3.

Hint 1: Can you modify the algorithm presented in the compendium for the k-center problem?

The key idea here is to apply the k-center algorithm to the set D. Then select the suppliers from F that are closest to the solution set for D.

To see that does indeed give us an 3-approxation algorithm we start by picking an arbitrary client $v \in D$. There could be a $d_i \in D'$ such that $d(v, d_i) \leq 2r^*$, where r^* is the optimal radius. This holds from the k-center problem we solved for D. Then we use the properties of the Euclidean distance, namely that: $d(v, f_i) \leq d(v, d_i) + d(d_i, f_i) \leq 2r^* + r^* = 3r^*$.

We can see that the distance from a chosen client to the nearest supplier, $d(d_i, f_i) \leq r^*$, must be less than or equal to the optimal radius due to the fact that it defines the solution. If it were bigger then r^* this would contradict r^*, which is defined by the longest distance from a supplier to a client in the minimised solution.

The proof now holds when there is an $d_i \in D$ with $d(v, d_i) \leq 2r^*$. Now we consider the case that no such d_i exists.

Here again we use the definitions from the k-center algorithm. We see that by the construction of the algorithm we have that for any two clients there is at most $2r^*$ distance between them otherwise r^* would not be the solution.

The algorithm more formally presented in sudocode:

Algorithm 2 A 3-approximation algorithm for the k-supplier problem.

```
D' ← k-center(k, D)
S ← ∅
while D' ≠ ∅ do
    i ∈ D'
    D' ← D'\{i}
    S ← S ∪ {arg min_{j∈F} d_ij}
return S
```

b Show that there cannot exist an approximation algorithm for the k-supplier problem with approximation factor less than 3 unless P = NP.

Hint 1: Take a look at the problem of finding a dominating set of k vertices.

Here I will try to reduce from the *dominating set problem* (DS) to the k-supplier problem (KS). Then showing that since DS is NP, an α-approximation algorithm A, with $\alpha < 3$, does not exists unless P = NP.

Let's consider the *dominating set problem*. This problem is NP-complete, and is defined roughly as: Given a

graph $G = (V, E)$ and $k \in \mathbb{N}$, we check if there exists a subset $S \subseteq V$ where $|S| = k$. Each vertex must either be in the set S or be adjacent to an element in S. The adjacency is defined to a given constant distance.

There exists a client, d_v, and supplier, f_v, for all vertices, v, in the graph, G: $\exists d_v \wedge f_v,\ \forall v \in V$. This gives us $2|V|$ clients and suppliers. The definition of the distances is quite simple, between any pair of suppliers or pair of clients, the distance is 2: $d(f_v, f_w) = d(d_v, d_w) = 2,\ \forall (v, w) \in V$. Then we state that the distance from a supplier-client pair is 1, and for neighbouring vertices in G: $d(d_v, f_v) = 1,\ \forall v \in V$ and $d(d_v, f_w) = 1,\ \forall (v, w) \in V \cap E$. All other distances is equal to 3.

The choice of distances 1, 2 and 3 is to uphold the Euclidean distance constraint in form of the triangle ratio.

For the proposed setup of the problem we either get a solution of 1 or 3.

Assuming that there exists an α-approximation algorithm A with $\alpha < 3$ for the DS. For the DS problem we are given G and k. The DS will check if there exists a set of k elements that fulfil the requirements. We reduced our k-supplier problem into a DS above, and can therefore now use algorithm A to solve it. If there exists a solution, to the dominating set, of value k then, the optimal value of the k-suppliers instance is 1. The algorithm A will then have an answer in the range: $\alpha \cdot 1 < 3$.

If the was not set for k, then the optimal value is 3. Then A will produce ≥ 3. We now know from the answer of the algorithm A that if a set of size k exists. However, this is not found in polynomial time, since the DS can't be solved in polynomial time. So, unless P = NP we can conclude that an α-approximation algorithm for this problem with $\alpha < 3$ does not exist.[3]

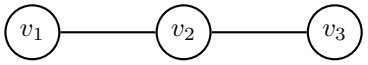


Figure 1: *Dominating set problem* graph G

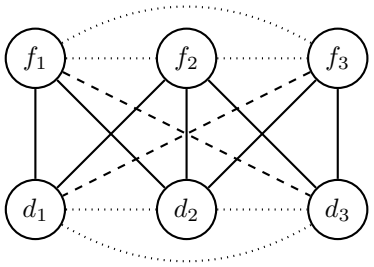


Figure 2: Illustration of k-supplier problem reduced from the *dominating set problem*. Distances: line: 1; dotted: 2; and dashed: 3. Based on source.[3]

Task 4

Let $G = (V, E)$ be a simple undirected graph. Let (E, I) be a set system with I being the family of subsets of E that induce a subgraph that is a *forest* in G.

a Show that the set system is an independence system.

"An independence system is a pair (E, S), where E is a non-empty set and S is a non-empty subset of the power set of E closed under inclusion: $A \in S$ and $B \subset A$ imply $B \in S$. The elements of S are called independent sets."[2] (page. 150).

We have that E is the set of edges in G. I is a set of subsets of E: $i \subseteq E,\ \forall i \in I$. So by the definition in the compendium (E, I) is an independence system.

b Show that the independence system is a matroid.

By theorem 5.1.2 (page. 151) in the compendium[2], we have that: "Let $G = (V, E)$ be a graph, and let S be the set of those subsets of E which are forests. Then (E, S) is a matroid."[2].

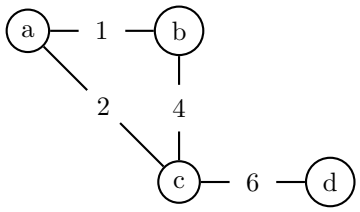
It's trivial to see how we map our problem onto this theorem. We have a graph G, and a set I that is a set of subsets of E which are a forest. Hence, (E, I) is a *matroid*.

I will also try and show it more explicitly from the more general definition of a matroid.

We have a ground set E which is not empty, $E \neq \emptyset$. The we have I which is not empty and contains the family of subsets of E that induce a subgraph that is a forest in G. Now I has to fulfil three properties:

- 1: $\emptyset \in I$; states that the empty set must be an independent set in I, which it trivially is.
- 2: $A \subseteq B$, $B \in I \Rightarrow A \in I$; this enforces the inheritance property. This hold, since if we i.e. remove a tree from the forest it's still a forest (or a tree, or the empty set which is part of I, as per rule 1) in G. Removing any edges either shrinks a tree of splits it into two new trees. Removing an edge can't create a cycle, hence I is still a set of forests in G.
- 3: $A, B \in I$, $|A| > |B| \Rightarrow \exists e \in A$ s.t. $B \cup \{e\} \in I$; this holds too. If A has a few more trees then B and we add one of these to B it is trivial that B is still a forest in G. Note: not all edges/trees in A might be added to B, since it might cause a cycle in B. However, if A sis bigger then B there exists at least on vertex v that is not touched by B that can extend B. Hence, this property holds true.

Since our set (E, I) fulfils all properties we have a matroid.



c Explicitly write out the matroid for the graph above. In other words: The matroid is (E, I), so how E and I are expressed by a, b, c, d?

First of we have E, the ground set of edges in the graph, G. This is simply:

$$E = \{e(a,b), e(a,c), e(b,c), e(c,d)\}$$

Where $e(v, w)$ is the edge from vertex v to vertex w.

Then there is I which I will try and represent in the same way:

$$\begin{aligned} I = &\{\emptyset, \{e(a,b)\}, \{e(a,c)\}, \{e(b,c)\}, \{e(c,d)\}\} \text{ Set of forests with one edge, and the empty set} \\ &\cup \{\{e(a,b), e(a,c)\}, \{e(a,b), e(b,c)\}, \{e(a,b), e(c,d)\}, \{e(a,c), e(b,c)\}, \\ &\quad \{e(a,c), e(c,d)\}, \{e(b,c), e(c,d)\}\} \text{ Set of forests with two edges} \\ &\cup \{\{e(a,b), e(a,c), e(c,d)\}, \{e(a,b), e(b,c), e(c,d)\}, \{e(a,c), e(b,c), e(c,d)\}\} \text{ Set of forests with three edges} \end{aligned}$$

I believe I is complete, but there is a non-insignificant chance I messed it up a bit, but I think it shows the gist of it at least.

d What is the partial solution T_i for $i = 1, \ldots, n$ of the greedy algorithm applied to the matroid (E, I)?

First we can define the greedy algorithm for a general independence system:

Algorithm 3 The general greedy algorithm for an independence system.

Ensure: (E, S) ▷ Is an independence system/set
Ensure: $E = \{e_1, \ldots, e_n\}$ with $w(e_1) \geq w(e_2) \geq \cdots \geq w(e_n)$ ▷ E is ordered according to weight
  $T \leftarrow \emptyset$
  while $\exists e_k \in E$ s.t. $T \cup \{e_k\} \in S$ **do**
    $T \leftarrow T \cup \{e_k\}$
  return T

We can then define our weight function as $w : E \to \mathbb{R}^+$. We want to find a maximal weight for an independence set T as follows:

$$w(T) = \sum_{e \in T} w(e)$$

Applying the algorithm to the graph above we get:
We order E by descending order of weight:

$$E = \{e(c,d), e(b,c), e(a,c), e(a,b)\}$$

I is as defined above. Then we begin picking the largest element in E that doesn't break our constraints when adding it to the solution set T.

T_0 is the empty set.
T_1 we add $e(c,d)$.
T_2 we add $e(b,c)$ and get $T_2 = \{e(c,d), e(b,c)\}$.
T_3 we add $e(a,c)$ and get $T_3 = \{e(c,d), e(b,c), e(a,c)\}$.
Now we have not $e_k \in E$ that can be added to T without breaking our constraints, hence we are done. The solution is illustrated in figure d and has a total weight of 12.

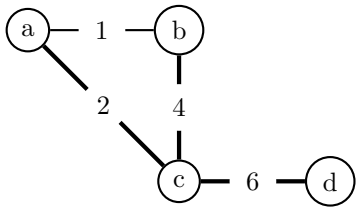


Figure 3: The solution to the greedy algorithm on the graph. The total weight is: 12. The subgraph solution is bolded out.

Task 5

You have a matroid M, but now you want to add a constraint on the number of elements you can have in the basis. If you require that you can have at most k elements in a basis, do you still have a matroid?

To explain this I will use linear algebra and vectors as examples, since I find this the most intuitive way to think about it.

First of we establish the fact that all bases of M is of the same size, b. Think of the three unit vectors (i, j and k, normally) used to describe 3D-space. These each have one direction along their respective axis of a distance 1. This means any vector in 3D-space can be produced by a linear combination of these, hence their are a basis of 3D-space. If we were to remove one of them, say i, we loose the ability to "move" along the x-axis and we can no longer describe any 3D-vector with the base set of just j and k. If our optimum lays in the non-zero realm of the x-axis it is now unreachable. So we need at least the three of them. If we add any other vector to the basis set, it can, as stated, be described by a linear combination of the others, hence we need exactly three (linearly independent) vectors in the basis.

This means that if we now look at the question about the k restriction on the number of elements in the basis. M will no longer be a matroid if we have k less than three in our example. This translates directly to all other equivalent definitions of matroids as well. By removing elements from the natural basis of the matroid, it losses the property to produce optimal solutions for the problem since the scope of the problem has been infringed upon, reducing the solution space.

Task 6

This task is taken from the exam TDT4125, spring 2018.

You are planning a travel route on a weighted, directed graph, where each edge is marked with one of several possible landscape types (mountains, forest, urban areas, etc.). You want to find a path that is as long as possible, but you want to visit each city (vertex) and each landscape type at most once. To ensure that you find a solution, you allow helicopter rides between cities. These have no length (weight), and no landscape type. You try a greedy solution, where you constantly add the longest edge that fits into your plan so far (possibly with several helicopter trips, to make it work). Give a lower bound on the degree of approximation of the greedy algorithm by describing the problem as the intersection of matroids.

Matroids described in the compendium can be assumed correct (valid), for any other matroids, the matroid properties must be proven.

Firstly I like to introduce some variables: $G = (V, E \cup H)$ is the weighted digraph described above. E is a set of pairs of weights and landscape types: $e_k \in E, e_k = (w_{ij}, l_{ij}), i, j \in V$. H is the edges from the helicopter.

Given that we can use the helicopter to travel between any pair of vertices the underlying graph is actually a weighted complete digraph, with weights equal to zero. This enables us to find a set of paths in the graph and connect them by the helicopter, creating a valid path. The problem then becomes more interesting. We can now define our first matroid, M_1. G is the weighted complete digraph obtained by the helicopter, but still having the edges with landscape. $M_1 = (E \cup H, I_1)$ is the forest matroid of G (much the same as in task 4). Since we are looking for a path (and a path is a valid forest), all our valid solutions, our solution space: I, is a subset of I_1.

Now we have generated all our solutions, but a few extra ones as well. We now introduce $M_2 = (E, I_2)$ to restrict our solution space. I_2 is the set of edges such that no edges has the same landscape.

$$I_2 = \{F \subseteq 2^E : e_{1l} \neq e_{2l}, \forall (e_1, e_2) \in F\}$$

To show that the matroid hold:

- 1: $\emptyset \in I_2$; states that the empty set must be an independent set in I_2, which it trivially is.
- 2: $A \subseteq B$, $B \in I_2 \Rightarrow A \in I_2$; this enforces the inheritance property. This hold, since if we remove an edge from the set it's still a valid set of edges. Removing an edge can't violate the constraints, since it removes a landscape, hence I_2 is still a set of edges in E.
- 3: $A, B \in I_2$, $|A| > |B| \Rightarrow \exists e \in A$ s.t. $B \cup \{e\} \in I_2$; this holds too. If A has a more edges then B and we add one of these to B it is trivial that B is still a set of edges in E. Note: not all edges in A might be added to B, since it might cause a repeat of landscape in B. However, if A contains all landscapes B has, it still has at least one landscape not in B by definition. Hence, this property holds true.

Now we need to enforce the fact that we want a path, given two matroids: $M_3 = (E \cup H, I_3)$ and $M_4 = (E \cup H, I_4)$.

If a set of edges is of size one, it's obviously a valid path. Otherwise, the degree of all vertices is 2, except for the start and end vertex, which has degree 1. More specifically, all vertices in the path has an in degree, $\delta^-(v)$, of 1 except the start vertex which has 0. Same with out degree, $\delta^+(v)$, where the end vertex is 0.

I_3 ensures that no vertex has more than one edge into it.

$$I_3 = \{F \subseteq 2^{E \cup H} : |F \cap \delta^-(v)| \leq 1, \forall v \in V\}$$

Showing that M_3 is a matroid:

- 1: $\emptyset \in I_3$; states that the empty set must be an independent set in I_3, which it trivially is.
- 2: $A \subseteq B$, $B \in I_3 \Rightarrow A \in I_3$; this enforces the inheritance property. This hold, since if we remove an edge from the set the in degree of one vertex is decreased from 1 to 0. Removing an edge can't violate the constraints. Hence I_3 is still a set of edges in E.
- 3: $A, B \in I_3$, $|A| > |B| \Rightarrow \exists e \in A$ s.t. $B \cup \{e\} \in I_3$; this holds too. If A has a more edges then B and we add one of these to B it is trivial that B is still a set of edges in E. Not all edges in A might be added to B though, since it might cause an in degree to be greater than 1 in B. However, if A contains one more element than B, A can't share more than $|B|$ vertices with B, hence there must be a vertex in A, who has in degree 0 in B, by definition. Hence, this property holds true.

I_4 ensures that no vertex has more than one edge out of it.

$$I_4 = \{F \subseteq 2^{E \cup H} : |F \cap \delta^+(v)| \leq 1, \forall v \in V\}$$

Showing that M_4 is a matroid:

- 1: $\emptyset \in I_4$; states that the empty set must be an independent set in I_4, which it trivially is.
- 2: $A \subseteq B$, $B \in I_4 \Rightarrow A \in I_4$; this enforces the inheritance property. This hold, since if we remove an edge from the set the out degree of one vertex is decreased from 1 to 0. Removing an edge can't violate the constraints. Hence I_4 is still a set of edges in E.
- 3: $A, B \in I_4$, $|A| > |B| \Rightarrow \exists e \in A$ s.t. $B \cup \{e\} \in I_4$; this holds too. If A has a more edges then B and we add one of these to B it is trivial that B is still a set of edges in E. Not all edges in A might be added to B though, since it might cause an out degree to be greater than 1 in B. However, if A contains one more element than B, A can't share more than $|B|$ vertices with B, hence there must be a vertex in A, who has out degree 0 in B, by definition. Hence, this property holds true.

I_3 and I_4 is inspired by a solution to a similar problem[1](Problem 3).

We now can combine the matroids given the intersection and get $M = (E \cup H, I), I = I_1 \cap I_2 \cap I_3 \cap I_4$. To recap: I_1 gives us all possible forests of the graph, which includes all possible valid path solutions. Then I_2 filters out all forests which has the same landscape more than once. I_3 and I_4 then reduced the solution space to only forests with trees without any branches, meaning a set of paths. Then we can get the maximum of this set. Since all paths in all the sets can be connected with the helicopter routes, forming one path, these also add no weight (length). Hence, we have a solution. Since we have used 4 matroids the approximation factor is: $\alpha = \frac{1}{k} = \frac{1}{4} = 0.25$[2](page. 165, Theorem 5.4.8).

Task 7

This task is taken from Jungnickel

Let $G = (V, E)$ be a connected graph that is not a tree. Prove that the subsets of E containing at most one cycle form a matroid over E. Would the same hold if we allowed a maximum of two cycles?

Since a tree is a undirected connected acyclic graph, the graph, $G = (V, E)$, must be a undirected connected cyclic graph. The statement in the task is that there is a independence system forming a matroid, $M = (E, I)$, over E given that the set I contains subsets of E with at most one cycle.

Let's look at the properties:

- 1: $\emptyset \in I$; states that the empty set must be an independent set in I, which it trivially is.
- 2: $A \subseteq B$, $B \in I \Rightarrow A \in I$; this enforces the inheritance property. This hold, since if we remove an edge from the set it's still a subgraph of G and a subset of E. If it had one cycle we might loose the cycle, but that's still a valid set. Hence, I is still a valid set of edges in E.
- 3: $A, B \in I$, $|A| > |B| \Rightarrow \exists e \in A$ s.t. $B \cup \{e\} \in I$; this holds too. If A has a more edges then B and we add one of these to B it is trivial that B is still a set of edges in E. Not all edges in A might be added to B though, since it might cause us to get two cycles in B. However, let's say that A contains one more element than B, and all elements in B is in A, hence forcing the choice of edge from A. For B to then form two cycles, A must have two cycles, but this can't be since $A \in I$. If A had an edge e that would form a second cycle in B it must then have a different vertex too, not in B, which then could be added instead. Hence, this property holds true.

We have now shown that this matroid does in fact exist.

We can have the other case of at most two cycles. As I will try and prove by the properties:

- 1: $\emptyset \in I$; states that the empty set must be an independent set in I, which it trivially is.

- 2: $A \subseteq B$, $B \in I \Rightarrow A \in I$; this enforces the inheritance property. This hold, since if we remove an edge from the set it's still a subgraph of G and a subset of E. If it had one or two cycle we might loose a cycle, but that's still a valid set. Hence, I is still a valid set of edges in E.

- 3: $A, B \in I$, $|A| > |B| \Rightarrow \exists e \in A$ s.t. $B \cup \{e\} \in I$; this holds too. If A has a more edges then B and we add one of these to B it is trivial that B is still a set of edges in E. Not all edges in A might be added to B though, since it might cause us to get three or more cycles in B. If B has one or two cycles and adding an edge would connect them such that we get three or more cycles. There does not exist a set A such that it forces that edge to be picked. I try to prove it by contradiction: let's say that A contains one more element than B, and all elements in B are in A, hence forcing the choice of edge from A. We say this edge can't be added to B. If it is added B would then form three or more cycles, for this to be the case A must have three or more cycles, since when adding the edge $B = A$, but this can't be since $A \in I$. If A has more elements we could always reduce A to the case of having just one more element than B by the second property. If A does not contain all elements of B then A has even more edges to pick from that are valid. Hence there exists an element in A such that we can add it to B. Hence, this property holds true.

References

[1] Prof. Eisenbrand. *École Polytechnique Fédérale de Lausanne*. URL: https://www.epfl.ch/labs/disopt/wp-content/uploads/2018/09/assignment09-solution.pdf. (accessed: 12.03.2023).

[2] NTNU. *Kompendium, Algoritmekonstruksjon*. NTNU, 2023.

[3] Williamson og Shmoys. *The Design of Approximation Algorithms - Answers to some of the exercises*. URL: https://personal.vu.nl/r.a.sitters/advancedalgorithms/AdvAlgExCh2.pdf. (accessed: 10.03.2023).