# AAD Aflevering 5

Andreas Hammer (qgk998), Anders Munkvad (tqs326), Johan Topp (fpg662)

January 2024

## Approximation Algorithms

### 35.1-1

Since the algorithm always adds 2 vertexes at each step if we an odd number of vertexes in the optimal solution, then we will always have a suboptimal solution.
An example of this would be a graph with $n$ vertexes and every vertex would have an edge into a center vertex $v'$ meaninig for every vertex $v_i \in V/v'$ there would be an edge $(v_i, v')$. Meaning we would have $n-1$ edges. No matter how the algorithm runs it will pick a random edge out of these and add $v'$ and another vertex $v_i$ to the vertex cover. This will always be a suboptimal solution because the optimal solution would trivially be just $v'$.

### 35.1-4

For a vertex cover we have to make sure that every edge has an endpoint in $C$. For a tree structure we can show that this problem has optimal substructure. If we add a node to $C$, and this is an optimal so far, then all children nodes are disjoint subgraphs, and if we can find optimal solutions to these subgraphs independently and achieve an optimal solution to the overall problem. This means we can just treat the children as the root of a new tree and find a vertex cover for it. But how do we choose an optimal solution. Since we can always treat a node with a parent that is in $C$ as a root, we just look at the problem from the perspective of the root of a tree. If the root has more than 1 child, then we need to resolve each of these edges to have one node in $C$, and the optimal way of doing this is adding the root to $C$, becuase either we would have to add the root or every children, but since children are $> 1$ then we add the root instead. If the root has 1 child, then it is best to add the child to $C$ and propogate the problem further. Since the all edges will have an enpoint in $C$ no matter if we choose the root or the 1 child, it is better to choose the child, since it might resolve future edges. If the root has no children then we don't add it, since there are no more edges to consider. Pseudocode can be seen in algorithm 1

This algorithm should always return an optimal vertex cover for a tree structure, in linear time, since we only visit each node once.

### 35.1-5

No it does not. Since the two are complements, then if a vertex cover for $G$ is found to be $C$ where $C$ has the size of $k$, then the solution to the clique problem, would be the maximum clique found in the complementary graph.
Assuming that $k$ is the minimum vertex cover, then the size of the maximum clique would be $n-k$, to contain all vertices not in $C$. And this does not only depend on $k$, but on the size of $n$, so as $n$ increases, the complexity will also increase, and therefore not be polynomial to a constant.

**Algorithm 1** VERT-COV-TREE

```
if already in C then
    for child in childset do
        VERT-COV-TREE(child)
else
    if child set is empty then return C
    else if child set has 1 element then
        child ← childset
        C ← C + child
        VERT-COV-TREE(child)
    else
        C ← C + current_node
        for child in childset do
            VERT-COV-TREE(child)
```

## 35.2-5

To show that a optimal tour never crosses itself we will show by contradiction. Suppose we have completed some part of an **optimal** tour, without crossing any edges. At some point we then try to add an edge $(w, z)$, that crosses another edge $(v, u)$. We can take the point in which these two edges cross each other and call it $c$.

Now we would like to make some definitions. We can use our new point $c$ to split the cost of the

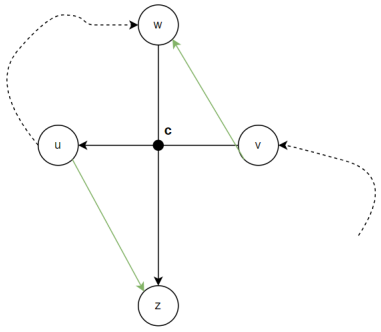


Figure 1: Illustration of 35.2-5

edges, since the cost is just the euclidian distance. $c(v, u) = |\bar{vc}| + |\bar{cu}|$ and $c(w, z) = |\bar{wc}| + |\bar{cz}|$. Meaning our total cost of these two edges is the same as the sum of the line segements. Now instead of the current path consider the two edges $(v, w)$ and $(u, z)$, which still make it so we visit all of the vertexes in this setup. We can use the triangle inequality as illustrated on figure 1, to say that $c(v, w) \leq |\bar{vc}| + |\bar{wc}|$ and that $c(u, z) \leq |\bar{cu}| + |\bar{cz}|$. Using this in combination with the defintion of the cost of the two original edges we get. $c(v, w) + c(u, z) \leq c(v, u) + c(w, z)$. If we vertexes to overlap, meaning the edges would need a euclidian distance of $> 0$ between them, we can change this inequality to be strict, since $c$ will never intersect with the two new edges we created. All of this means we can create a path of less cost, while still visiting the same vertexes, by removing the edges crossing each other. This contradicts our initial assumption that the path was optimal, and shows that an optimal solution can never have edges crossing each other because we can always find a shorter path.

### 35.3-3

We want to make the running time of the algorithm to be $O(\sum_{s \in \mathcal{F}} |S|)$. In order to achieve this, we slightly modify the algorithm given by the book. Firstly, we select $S$ that maximises $|S \bigcap U|$, just as they do in the book. After we have added this $S$ to $E$ we consider another $S$ that we call $S_x$. Now we want to find the $S_x$ that maximises $|S_x \setminus (e \bigcup U_y)|$. We do this, as we want to pick the largest $S$ that has the most elements that is not already in $e$. Since, in the worst case, we have to add all of the elements in $S$, hence $O(|S|)$. Once this has been done, since we now select a $S$ that maximises $|S_x \setminus (e \bigcup U_y)|$, we now only add certain elements already missing from $e$. By doing this we get the worst case running time of $O(\sum_{s \in \mathcal{F}} |S|)$, since we make sure to either add the entire $S$ or single elements from it (those missing from $e$).

**Algorithm 2** GSC

$U = X$
$e = \emptyset$
**while** $U \neq \emptyset$ **do**
    `Select` $S$ `that maximizes` $|S \bigcap U|$
    $U_y = U \setminus S$
    $e = e \cup \{S\}$
    `Select` $S_x$ `that maximizes` $|S_x \setminus (e \bigcup U_y)|$
    $U_x = U_y \setminus S_x$
    $e = e \cup \{S_x\}$
**return** $e$

### 35.4-2

In order to give a randomized 2-approximation algorithm for the MAX-CNF satisfiability problem, we can simply assign a truth values (being either true or false) to each literal at random. Hence, each literal has a 50% chance of being either true or false. If we consider a given clause with $n$ literals, the probability of this given clause not being satisfied by the random assignement of truth values is $\frac{1}{2^n}$ (because we assign literals randomly, hence $\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot ... \frac{1}{k}$, and each clause must be assigned incorrectly in order for the clause to fail). Because of this, the probability that the given clause is satisfied will be at least $1 - \frac{1}{2^n}$ and since we have that $n \geq 1$ since each clause has at least 1 literal, we have that $1 - \frac{1}{2} = \frac{1}{2}$ showing that it is a 2-approximation.

### 35.4-3

If we are given an unweighted undirected graph $G = (V, E)$ and define a cut $(S, V - S)$ and the weight of the cut as the number of edges crossing the cut and we want to find the maximum weight. We want to show that it is a randomized 2-approximation algorithm, if we for each vertex $v$, we randomly and independently place $v$ in $S$ with probability $\frac{1}{2}$ and in $V - S$ with probability $\frac{1}{2}$.

In order to show this, we consider what needs to happen in order for en edge, e.g. $(u, v)$ to be in the cut, namely one of the two scenarios:

- Vertex $u$ is in $S$ and $v$ is in $V - S$.
- Vertex $u$ is in $V - S$ and $v$ is in $S$.

Since each vertex has a probability of $\frac{1}{2}$ to be in either set, each of the two scenarios has $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$. Hence, the total probability that an edge is in the cut is:

$$\frac{1}{4} + \frac{1}{4} = \frac{1}{2}$$

Since we have $E$ edges, the expected weight of the cut would be $\frac{|E|}{2}$, since the expected number of edges in the cut is half of the edges. If we now consider the optimal cut, which we denote as $OPT$, we want to show that our solution is at least half of $OPT$, namely:

$$\frac{|E|}{2} \geq \frac{OPT}{2}$$

Since we have that the optimal solution cannot have more edges than there is in the graph, namely $OPT \geq |E|$, the above inequality holds, showing that it is indeed a 2-approximation algorithm.

### 35.5-2

**Base case**
The base case is taken with i = 1, since for i = 0, there are no integers to sum.
$L_0 = 0$
$L_1 = 0, x_i$, from `Merge-Lists`.
In the case that $t < x_i$ then line 5 of the algorithm will remove $x_i$, resulting in $z = y = 0$.
In the case that $x_i < t$ then the algorithm will return $z = x_i = y$ in which case 35.24 holds, since $\frac{y}{1+\delta} \leq z \leq y$ since $z$ can be substituted with $y$ resulting in $\frac{y}{1+\delta} \leq y \leq y$ and $0 < \delta < 1$

**Induction step**
The optimal solution $y$ either belongs to the set $P_i = P_{i-1} \cup (P_{i-1} + x_i)$. (35.23). In the case that $y \in P_{i-1}$ then by proof of induction, $y$ has already been found in $P_{i-1}$. In the case that $y \in (P_{i-1} + x_i)$, then it either belongs to $L_i$ after it is trimmed or it belongs to the discarded part of $L_i$.
Assuming that for a given $k$ the induction holds, then the case where $\frac{y}{(1+\frac{\epsilon}{2n})^k} \leq z \leq y$ holds.
For $k+1$ then we need to prove that $\frac{y}{(1+\frac{\epsilon}{2n})^{k+1}} \leq z \leq y$.
First we rewrite $\frac{y}{(1+\frac{\epsilon}{2n})^{k+1}} = \frac{y}{(1+\frac{\epsilon}{2n})^k \cdot (1+\frac{\epsilon}{2n})}$ resulting in

$$\frac{y}{(1+\frac{\epsilon}{2n})^k \cdot (1+\frac{\epsilon}{2n})} \leq z \leq y$$

Multiplying with $1 + \frac{\epsilon}{2n}$ on both sides of the inequality:

$$\frac{y}{(1+\frac{\epsilon}{2n})^k} \leq (1 + \frac{\epsilon}{2n}) \cdot z \leq y$$

$P_{i+1}$ contains $L_{i+1}$ before the trim. So either $z$ is in the untrimmed set, in which case the inequality holds.
Otherwise it must be in the part that was removed during the trimming. If the correct value for $z_{k+1}$ is a part of the removed set, then it will be represented by a value in $z_k$ which is at most $(1 - \delta) \cdot z_k$ less then the actual value of $z$, otherwise it would not have been trimmed, Therefore the inequality holds.

# 1 Andreas Hammer

## 1.1 Approximation Algorithms

- minimization and maximization problems approx.
- vertex cover example
- Set Cover Greedy approach
- Proof that this approach is O(log n)-approx alg.

# 2 Anders Munkvad

## 2.1 Approximation Algorithms

- Min/max approximation problems.
- Approximation Ratio
- Example: Vertex cover
- Example: MAX-CNF (or 3-CNF-SAT)

# 3 Johan Topp

- Optimization problem type max(c/C*) vs min(c*/c)
- Run-time of exact algorithms
- vertex cover
- c-approximate (2 for vertex. show)
  - reflect on TSP, approx-TSP, polynomial time 2-approx, vs exact exponential algorithm
- show Greedy-Vertex-Cover
  - Alternatively show TSP, using Euler tour. Depending on difficulty of the two.