# Algoritmekonstruksjon TDT4125 - Assignment 4

Ola Kristoffer Hoff

24th April, 2023

## Task 1

We will now deal with the *MAX-SAT* problem based on the following expression

$$(x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2).$$

### a Is it possible to satisfy all the clauses?

No, as we can see, all possible combinations of the to variables are in the clauses, hence on of them will be false.

### b Which assignment maximizes the number of satisfied clauses?

All assignments are equal in this case. There will always be one, and only one, clause that is false. This can easily be check if you write it done as a truth table.

If you set $x_1$ to true the first two clauses are true, if you set it to false, the last two are true. Then we see that in both cases the two remaining clauses have an opposite $x_2$ meaning that one will be true and one will be false. This gives us in all cases three true clauses and one false clause.

## Task 2

*This task is taken from Williamson & Shmoys - exercise 5.1*

In the *k-cut* problem, we are given an undirected weighted graph $G = (V, E)$ with non-negative weights $w_{ij}$ for all $(i, j) \in E$. The goal is to divide the set of vertices $V$ into $k$ disjoint sets $V_1, V_2, \ldots, V_k$ so that we maximize the sum of the edge weights of the edges that go between different sets of vertices. Give a randomized $\frac{k-1}{k}$-approximation algorithm for the $k$ cut problem.

Here we can extend the proof of the $MAXCUT$ from *Williamson & Shmoys* chapter 5.1:

"Consider a random variable $X_{ij}$ that is 1 if the edge $(i, j)$ is in the cut, and 0 otherwise. Let $Z$ be the random variable equal to the total weight of edges in the cut, so that $Z = \sum_{(i,j)\in E} w_{ij} X_{ij}$. Let $OPT$ denote the optimal value of the maximum cut instance. Then, as before, by linearity of expectation and the definition of expectation of a 0−1 random variable, we get that"[2](page. 248).

$$E[Z] = \sum_{(i,j)\in E} w_{ij} E[X_{ij}] = \sum_{(i,j)\in E} w_{ij} Pr[\text{Edge } (i,j) \text{ in cut}].$$

In our case we see that the probability of an edge being in the cut can be found by finding the opposite first. If we assign $v_j$ to the same set as $v_i$, $(i, j)$ is not in the cut, since we have $k$ sets there is a $\frac{k-1}{k}$ chance of picking a set, at random, that includes the edge in the cut.

Hence we get a $\frac{k-1}{k}$-approximation algorithm.

## Task 3

*This task is taken from Williamson  Shmoys - exercise 5.3*

In the maximum directed cut problem, we are given a weighted directed graph $G = (V, E)$ with non-negative weights. The goal is to divide $V$ into two disjoint sets $U$ and $W = VU$ so that the sum of the weights of the arcs going from a vertex in $U$ to $W$ is maximized. Give a randomized $\frac{1}{4}$-approximation algorithm for this problem.

This is an easy extension of the undirected *MAX CUT* problem. Using the same proof as in the previous section, b.

We can see that the linearity is the same, hence we just need to modify the probability of the edge being in the cut. Since we are splitting the set $V$ into to sets, there is a $\frac{1}{2}$ probability that the edge is included in the (undirected) cut. Then the edge either goes from $U$ to $W$ or vice versa. This gives again a $\frac{1}{2}$ probability of the edge going the correct direction, and being counted in the cut.

Hence, we get $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$-approximation algorithm.

## Task 4

Derandomize the simple randomized algorithm for *MAX CUT* from Section 5.1 (p. K251). How do you choose whether a vertex should be included in $U$ or $W$?

So for this we use the *method of conditional expectations*: "This technique for derandomizing algorithms works with a wide variety of randomized algorithms in which variables are set independently and the conditional expectations are polynomial-time computable."[2](page. 256).

So following the proof of chapter 5.2 in the "Williamson & Shmoys"[2], we see that all we need to change is the choice of which set to include the vertex in for the proof to hold, and that it has to be polynomial-time computable.

In this instance we can use a very similar and simple method of choice. We start with $U = W = \emptyset$. Then we add $v_i$ to $U$ if $w(N(v_i) \cap W) \geq w(N(v_i) \cap U)$, otherwise we add $v_i$ to $W$. In words: add $v_i$ to the set that in this instance increases the weight of the solution the most. Since this is always equal to or better than the random choice we inherit the approximation guarantee from the randomized algorithm.

## Task 5

At many fitness centres, swimming pools, museums, etc. you can buy both day tickets and annual tickets. What is worth buying depends on how many times you visit the place during a year, which you don't necessarily know until the end of the year. For each visit, you therefore have to decide whether you want to buy an annual ticket or continue to buy one-time tickets.

We can model this as an online problem, where the input is a stream of numbers (in ascending order) indicating a visit on the given day. For each number, you have to make a decision whether to buy a one-time ticket, buy an annual pass or use an annual pass you have bought previously. Let $p_1$ and $p_{365}$ be the price of a one-time ticket and an annual ticket, respectively. We want to minimize spending. Describe an algorithm that is *strictly 2-competitive* for this problem.

Here we can follow a simple set of rules. These we derive from what the offline algorithm would do. Here I use $N$ as the total number of visits that will occur.

So if $N > \frac{p_{365}}{p_1}$ then we reduce the cost by buying one-time tickets. On the other hand if $N < \frac{p_{365}}{p_1}$ we reduce cost by buying the annual ticket. So the offline algorithm would have a cost: $c \leq p_{365}$. For us to construct an algorithm that guarantees that we will spend no more than twice the amount of the offline algorithm we do as follows:

Buy up to $n$ one-time tickets until we have: $n = \frac{p_{365}}{p_1}$. Then we buy a annual ticket. If we run our three scenarios now:

- $N < \frac{p_{365}}{p_1}$: Here the offline would buy one-time tickets, and so does the online algorithm.
- $N = \frac{p_{365}}{p_1}$: Here the cost of buying just one-time tickets and an annual ticket is the same. So the offline and online algorithms get the same cost.
- $N > \frac{p_{365}}{p_1}$: Here the offline would buy the annual ticket for $p_{365}$, the offline would buy on-time tickets until it has paid as much for them as it costs for an annual ticket, at which point it buys an annual ticket. Here the online algorithm costs $2p_{365}$.

As we can see, the online alogrithm does optimaly in the first two cases, then it is *strictly 2-competitive* in the last case.

## Task 6

*This task is taken from "An Introduction to Online Computation: Determinism, Randomization, Advice" (Komm)*

For the paging problem, show that any algorithm that is allowed to swap out an arbitrary number of pages in the cache in each iteration can be replaced by an algorithm that can swap out only one page in each iteration, without increasing the number of pages which in total is swapped out for each input.

So this is fairly intuitive. Since we get one new page request in each iteration we would at most need to swap one page to satisfy the new page. If we swapped $k$ arbitrarily number of pages in iteration $i$ we could have spaced them out through iteration $i$ to $i + k - 1$. This would give the same results. Since the request rate is lower or equal to the swapping rate in both cases. Here's a loose analogy: it's in essence the same as doing all the prep work for your cooking before you start or right before you need it. You still do the same amount of prep work in total.

## Task 7

*This task is taken from "An Introduction to Online Computation: Determinism, Randomization, Advice" (Komm)*

In real-world situations, pages in memory are more likely to be requested the closer they are to the previous page that was requested. Based on this, we create the algorithm **Local**. With a replacement, this removes the page in the cache that is furthest away from the requested page. Is **Local** *competitive* for the paging problem?

I would say that it may be alright in some instances. However, I can think of a common problem for it. So this focuses on the phenomenon of *spatial locality*, there are two type of locality, the other one is *temporal locality*. The latter one states that pages used recently are likely to be used again. Now imagine a loop here we have great *spatial locality* since the we are using the same nearby data multiple times, but we also have *temporal locality* since we are using the same data again and not just nearby data. Now if we have some calculations in the loop for some maths it's all good and well. However, let's say it would be better to put the maths in a function, to have cleaner code. Now all of a sudden the maths are far away from the loop, page wise, so we evict the logic in the function, so every time the function is called we get a page fault. This has horrible utilisation of *temporal locality*. Thus, I would say, it's not a good contender. Since it is bad for function calls, which is one of the most common occurences in code.

## Task 8

*This task is taken from "An Introduction to Online Computation: Determinism, Randomization, Advice" (Komm)*

Show that $FWF$ is a *marking algorithm.*

The *Flush when Full* algorithm, "Empty the entire cache when a page fault occurs" (Lecture 12), is almost identical to the definition of the *marking algorithm*, which is (Lecture 12):

- Mark all slots of the cache.
- Each new page that is accessed is marked whether it was already in the cache or it was brought due to a page fault.
- When a page is brought to the cache due to a fault, it is placed at **some** unmarked slot in the cache.
- If all slots in the cache are marked when a page fault occurs, unmark all slots in the cache.

So since we mark all slots, we clear the cache when a page fault occurs for the first time in both algorithms. Having an empty cache is essentially the same as an unmarked cache, we get $n$, size of cache, fault before the cache is full/fully marked. Then again, when some page fault occurs we empty/unmark it again.

Hence, given the almost one-to-one correlation of the algorithms it's easy to see that the $FWF$-algorithm is a *marking algorithm.*

## Task 9

*This task is taken from Parameterized Algorithms - exercise 2.2*

Give an example of a feedback arc set $F$ in a tournament $G$, such that $G \circledast F$ contains a cycle/circuit.

*A feedback arc set is a set of edges, such that if we remove all the edges in the set from the graph, then there are no more cycles in the graph, but this does not apply to any subsets of this set.*

I do believe that the task is worded a bit misleading (or I am just confused). The definition of the *feedback arc set* given is, from what I can tell, the definition of the *inclusive-wise minimal feedback arc set.*

The definition of a *feedback arc set* is given as: "A feedback vertex set of a simple directed graph $G$ is a set of vertices whose removal makes $G$ acyclic; a feedback vertex set contains at least one vertex of every cycle in $G$. The term feedback vertex set also appears as essential set in the literature. A feedback vertex set $S$ is minimal if no proper subset of $S$ is a feedback vertex set."[1].

Whilst the lemma in the compendium states: "Let $G$ be a directed graph and $F$ be a subset of $E(G)$. Then $F$ is an *inclusion-wise minimal feedback arc set* of $G$ if and only if $F$ is an *inclusion-wise minimal set* of edges such that $G \circledast F$ is an acyclic directed graph."[2](page. 321, Lemma 2.7).

Note that in the lemma we have to have a *inclusion-wise minimal feedback arc set.* This means that for us to find a set $F$ that is a feedback arc set in $G$, it cannot be *inclusion-wise minimal feedback arc set.* This meaning it must contain at least two elements of one of the cycles. This means that it by definition has a subset that too is a feedback arc set.

Thus, I have understood the task as follows: show an example of a set $F$ which is not minimal, but still a feedback arc set. Which results in an acyclic graph if we remove the edges, but not if we flip the edges of $F$.

In figure 1 I have illustrated an example. By removing all edges in $F$ we get a directed acyclic graph. Thus, $F$ is by definition a valid feedback arc set in the tournament $G$.

We can also see that it does not give an acyclic graph from $G \circledast F$. Since $F$ contains a cycle itself, the resulting graph must also have the cycle.

Note: here we have the case that edge $(a, b)$ could be removed to get a sub set that is still a feedback arc set, in fact it would be minimal as well. However, this is as I discussed above what I understood the task to be.

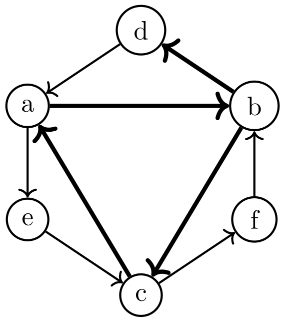

Figure 1: A graph $G$ containing a feedback arc set $F$, such that $G \circledast F$ contains a cycle. $F$ is highlighted in bold.

## Task 10

*This task is taken from Parameterized Algorithms - exercise 2.4*

In the *point line cover* problem, we are given a set of $n$ points in a plane and an integer number of lines $k$. The goal is to determine whether there is a set consisting of $k$ lines in the plane which in sum intersect all of the $n$ points. Find a kernel for this problem consisting of $O(k^2)$ points.

If the number of points is bounded by $O(k^2)$ we can remove all lines that has $k$ or more points, then decrease $k$ by 1. If we do this $k$ times we have removed $\geq k^2$ points. Since a line is defined by two points, if there at any point should be $n_i \leq 2(k - i)$ (points left at iteration $i$ less than two times the lines) it's by definition solvable.

Hence, if there is no line with $k$ or more points on it and there are more than $2k$ points left, the solution is no. Otherwise yes.

## References

[1] Ali Baharev; Hermann Schichl; Arnold Neumaier; Tobias Achterberg. *An exact method for the minumum feedback arc set problem.* URL: `https://www.mat.univie.ac.at/~herman/fwf-P27891-N32/minimum_feedback_arc_set.pdf`. (accessed: 24.04.2023).

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